BELIEFS: STATE UNCERTAINTY

AA228/CS238 Decision Making Under Uncertainty[1]

Robert Moss
Stanford University

mossr@cs.stanford.edu

November 15, 2023

[1] Mykel J. Kochenderfer, Tim A. Wheeler, and Kyle H. Wray. *Algorithms for Decision Making*. MIT Press, 2022.

Content outline

- Introduction to POMDPs
- Belief representations
- Algorithms to update beliefs (discrete/continuous)
- Interactive Julia notebooks (`Pluto.jl`):
  - Crying baby POMDP (from scratch)
  - Particle filtering
  - Kalman filtering
  - Bonus (linked): Crying baby POMDP (using `POMDPs.jl`)
  - ▶ `https://github.com/JuliaAcademy/Decision-Making-Under-Uncertainty#2-pomdps-partially-observable-markov-decision-processes`

POMDPs and Beliefs

- A POMDP[2] is an MDP with *state uncertainty*

[2]Partially observable Markov decision process. "Partially observable" is key in understanding beliefs.

POMDPs and Beliefs

- A POMDP[2] is an MDP with *state uncertainty*

$$\text{MDP: } \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$$
$$\text{POMDP: } \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$$

[2] Partially observable Markov decision process. "Partially observable" is key in understanding beliefs.

POMDPs and Beliefs

- A POMDP[2] is an MDP with *state uncertainty*

$$\text{MDP: } \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$$
$$\text{POMDP: } \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$$

- The agent receives an *observation* of the current state rather than the true state (potentially imperfect observations)

[2] Partially observable Markov decision process. "Partially observable" is key in understanding beliefs.

POMDPs and Beliefs

- A POMDP[2] is an MDP with *state uncertainty*

$$\text{MDP: } \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$$
$$\text{POMDP: } \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$$

- The agent receives an *observation* of the current state rather than the true state (potentially imperfect observations)
- Using past observations, the agent builds a *belief* of their underlying state

[2] Partially observable Markov decision process. "Partially observable" is key in understanding beliefs.

POMDPs and Beliefs

- A POMDP[2] is an MDP with *state uncertainty*

$$\text{MDP: } \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$$
$$\text{POMDP: } \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$$

- The agent receives an *observation* of the current state rather than the true state (potentially imperfect observations)
- Using past observations, the agent builds a *belief* of their underlying state
  - Which is a probability distribution over true states

[2] Partially observable Markov decision process. "Partially observable" is key in understanding beliefs.

POMDPs and Beliefs

- A POMDP[2] is an MDP with *state uncertainty*

$$\text{MDP: } \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$$
$$\text{POMDP: } \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$$

- The agent receives an *observation* of the current state rather than the true state (potentially imperfect observations)
- Using past observations, the agent builds a *belief* of their underlying state
  - Which is a probability distribution over true states
- Remember, a POMDP is a *problem formulation* and not an *algorithm*

[2] Partially observable Markov decision process. "Partially observable" is key in understanding beliefs.

POMDPs and Beliefs

- A POMDP[2] is an MDP with *state uncertainty*

$$\text{MDP: } \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$$
$$\text{POMDP: } \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, \gamma \rangle$$

- The agent receives an *observation* of the current state rather than the true state (potentially imperfect observations)
- Using past observations, the agent builds a *belief* of their underlying state
  - Which is a probability distribution over true states
- Remember, a POMDP is a *problem formulation* and not an *algorithm*
  - A POMDP formulation enables the use of solution methods, i.e. algorithms.

[2] Partially observable Markov decision process. "Partially observable" is key in understanding beliefs.

Observation Space

- The agent receives an observation o, which belongs to some *observation space* $\mathcal{O}$
- The probability of observing o given action a and next state s' is: $O(o \mid a, s')$
  - If $\mathcal{O}$ is continuous, then $O(o \mid a, s')$ is a probability density

Dynamic Decision Network for POMDPs

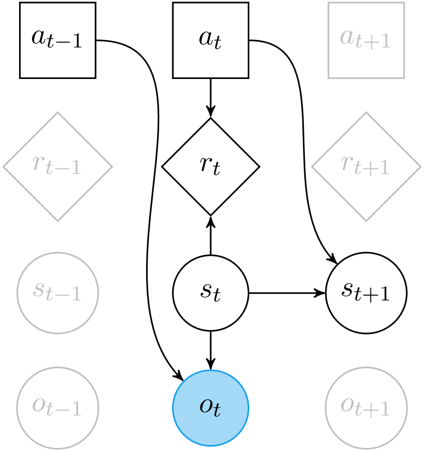


Figure: A dynamic decision network for the POMDP problem formulation.

Belief Representation

Beliefs can be represented in different ways:

[3] A probability mass is assigned to each discrete category.

Belief Representation

Beliefs can be represented in different ways:

- **Parametric**: The belief distribution is represented by a set of parameters for a fixed distribution family
  - E.g., Categorical distribution[3] or multivariate normal (Gaussian) distribution

[3] A probability mass is assigned to each discrete category.

Belief Representation

Beliefs can be represented in different ways:

- **Parametric**: The belief distribution is represented by a set of parameters for a fixed distribution family
  - E.g., Categorical distribution[3] or multivariate normal (Gaussian) distribution
- **Non-parametric**: The belief distribution is represented by particles (or points sampled from the state space)

[3] A probability mass is assigned to each discrete category.

Belief Representation

Beliefs can be represented in different ways:

- **Parametric**: The belief distribution is represented by a set of parameters for a fixed distribution family
  - E.g., Categorical distribution[3] or multivariate normal (Gaussian) distribution
- **Non-parametric**: The belief distribution is represented by particles (or points sampled from the state space)

Depending on the representation, different algorithms can be used to update beliefs.

[3] A probability mass is assigned to each discrete category.

Algorithms for Updating Beliefs

Various algorithms can update the current belief:

[4] Meaning we arrive at an analytical solution without approximations.

Algorithms for Updating Beliefs

Various algorithms can update the current belief:

- If the state space is *discrete* (or certain linear Gaussian assumptions are met), then we can perform *exact belief updates*:[4]
  - Recursive Bayesian estimation (i.e. discrete state filter)
  - Kalman filter

[4] Meaning we arrive at an analytical solution without approximations.

Algorithms for Updating Beliefs

Various algorithms can update the current belief:

- If the state space is *discrete* (or certain linear Gaussian assumptions are met), then we can perform *exact belief updates*:[4]
  - Recursive Bayesian estimation (i.e. discrete state filter)
  - Kalman filter
- Otherwise, we can use approximations based on:

[4] Meaning we arrive at an analytical solution without approximations.

Algorithms for Updating Beliefs

Various algorithms can update the current belief:

- If the state space is *discrete* (or certain linear Gaussian assumptions are met), then we can perform *exact belief updates*:[4]
  - Recursive Bayesian estimation (i.e. discrete state filter)
  - Kalman filter
- Otherwise, we can use approximations based on:
  - Linearization:
    - Extended Kalman filter (EKF)
    - Unscented Kalman filter (UFK)

[4] Meaning we arrive at an analytical solution without approximations.

Algorithms for Updating Beliefs

Various algorithms can update the current belief:

- If the state space is *discrete* (or certain linear Gaussian assumptions are met), then we can perform *exact belief updates*:[4]
  - Recursive Bayesian estimation (i.e. discrete state filter)
  - Kalman filter
- Otherwise, we can use approximations based on:
  - Linearization:
    - Extended Kalman filter (EKF)
    - Unscented Kalman filter (UFK)
  - Sampling:
    - Particle filter
    - Particle filter with rejection
    - Injection particle filter
    - Adaptive injection particle filter

[4] Meaning we arrive at an analytical solution without approximations.

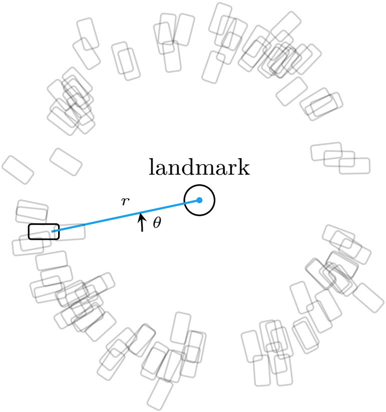
Belief Initialization

Before any actions or observations, we start with an initial belief distribution

- We can encode prior knowledge in the initial distribution
- Generally want to use diffuse (i.e. spread out) initial distributions to avoid over confidence in the absence of information
  - In non-parametric representations, a diffuse initial prior may cause difficulties
  - Thus, we may wait until an informative observation is make to initialize beliefs

Example: Landmark Belief Initialization

Figure: Localization of an autonomous car using a landmark (Example 19.1).



Making a range r and bearing θ observation, we initialize our belief around the landmark.

Belief Inference

- To infer the unknown belief distribution, we use *recursive Bayesian estimation*
  - Updates belief estimate recursively over time
  - Markov assumption: Only requires the current state, action, and observation
- Let $b(s)$ represent the probability[5] assigned to state s
  - A particular belief b belongs to a *belief space* $\mathcal{B}$ (containing all possible beliefs)
- For finite state and observation spaces, we can use a *discrete state filter* to perform exact inference

[5] or probability density for continuous state spaces

Belief Vector

- In the finite state case, we can represent beliefs using a categorical distribution[6]
  - Represented as a *belief vector* $\mathbf{b}$ of length $|\mathcal{S}|$, therefore $\mathcal{B} \subset \mathbb{R}^{|\mathcal{S}|}$
  - Sometimes $\mathcal{B}$ is referred to as a *probability simplex* or *belief simplex*[7]
- The belief vector $\mathbf{b}$ must be strictly non-negative and sum to one:

$$b(s) \geq 0 \text{ for all } s \in \mathcal{S} \qquad \sum_s b(s) = 1$$

- In vector notation:

$$\mathbf{b} \geq \mathbf{0} \qquad \mathbf{1}^\top \mathbf{b} = 1$$

- In Julia syntax:

```
all(b .≥ 0) && sum(b) ≈ 1
```

[6] A probability mass is assigned to each discrete state.

[7] Simplex being the generalization of a triangle to arbitrary dimensions.

Discrete State Filter: Updating Beliefs

A *filter* is a process that remove noise from data.[8]

Due to the independece assumptions, if an agent with belief b takes an action a and receives an observation o, then the new belief b' becomes:[9]

$$
\begin{aligned}
b'(s') &= P(s' \mid b, a, o) \\
&\propto P(o \mid b, a, s')P(s' \mid b, a) && \text{(Bayes' rule)} \\
&\propto O(o \mid a, s')P(s' \mid b, a) && \text{(observation definition)} \\
&\propto O(o \mid a, s')\sum_{s} P(s' \mid b, a, s)P(s \mid b, a) && \text{(law of total probability)} \\
&\propto O(o \mid a, s')\sum_{s} T(s' \mid s, a)b(s) && \text{(state transition model)}
\end{aligned}
$$

[8] Often used in signal processing, effectively "filtering" out the noise.

[9] For finite/discrete state and observation spaces.

Updating Beliefs: Derivation Explained

$$b'(s') = P(s' \mid b, a, o) \qquad \text{(probability of being in state } s'\text{)}$$

Exact belief updating:[10] $\quad b'(s') \propto O(o \mid a, s') \sum_s T(s' \mid s, a) b(s)$

[10] Then normalize so beliefs sum to one.

Updating Beliefs: Derivation Explained

$$\begin{aligned} b'(s') &= P(s' \mid b, a, o) && \text{(probability of being in state } s'\text{)} \\ &\propto P(o \mid b, a, s')P(s' \mid b, a) && \text{(Bayes' rule, dropping normalization)} \end{aligned}$$

Exact belief updating:[10] $b'(s') \propto O(o \mid a, s') \sum_{s} T(s' \mid s, a) b(s)$

[10] Then normalize so beliefs sum to one.

Updating Beliefs: Derivation Explained

$$
\begin{aligned}
b'(s') &= P(s' \mid b, a, o) && \text{(probability of being in state } s'\text{)} \\
&\propto P(o \mid b, a, s')P(s' \mid b, a) && \text{(Bayes' rule, dropping normalization)} \\
&\propto O(o \mid a, s')P(s' \mid b, a) && \text{(observation model def., } o \text{ independent of } b\text{)}
\end{aligned}
$$

Exact belief updating:[10] $b'(s') \propto O(o \mid a, s') \sum_s T(s' \mid s, a) b(s)$

[10] Then normalize so beliefs sum to one.

Updating Beliefs: Derivation Explained

$$\begin{aligned}
b'(s') &= P(s' \mid b, a, o) && \text{(probability of being in state } s'\text{)} \\
&\propto P(o \mid b, a, s')P(s' \mid b, a) && \text{(Bayes' rule, dropping normalization)} \\
&\propto O(o \mid a, s')P(s' \mid b, a) && \text{(observation model def., } o \text{ independent of } b\text{)} \\
&\propto O(o \mid a, s')\sum_{s} P(s' \mid b, a, s)P(s \mid b, a) && \text{(law of total probability)}
\end{aligned}$$

Exact belief updating:[10] $b'(s') \propto O(o \mid a, s')\sum_{s} T(s' \mid s, a)b(s)$

[10] Then normalize so beliefs sum to one.

Updating Beliefs: Derivation Explained

$$
\begin{aligned}
b'(s') &= P(s' \mid b, a, o) && \text{(probability of being in state } s'\text{)} \\
&\propto P(o \mid b, a, s')P(s' \mid b, a) && \text{(Bayes' rule, dropping normalization)} \\
&\propto O(o \mid a, s')P(s' \mid b, a) && \text{(observation model def., } o \text{ independent of } b\text{)} \\
&\propto O(o \mid a, s') \sum_s P(s' \mid b, a, s)P(s \mid b, a) && \text{(law of total probability)} \\
&\propto O(o \mid a, s') \sum_s T(s' \mid s, a)b(s) && \text{(state transition model, belief def.)}
\end{aligned}
$$

Exact belief updating:[10] $b'(s') \propto O(o \mid a, s') \sum_s T(s' \mid s, a)b(s)$

[10] Then normalize so beliefs sum to one.

Example: Crying Baby Problem

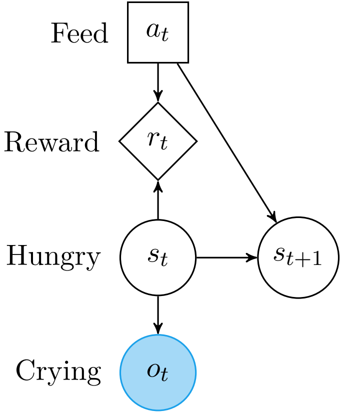


Figure: The crying baby POMDP.

- A simple POMDP with 2 states, 3 actions, and 2 observations:

$$\mathcal{S} = \{\text{hungry}, \text{sated}\}$$
$$\mathcal{A} = \{\text{feed}, \text{sing}, \text{ignore}\}$$
$$\mathcal{O} = \{\text{crying}, \text{quiet}\}$$

- See Pluto notebook:
  - crying_baby_problem.jl

Particle Filter

- *Particle filters* represent the belief states as a collection of states.
- Each state in the approximated belief is called a *particle*.
- Useful in problems with large discrete states spaces or continuous problems not well approximated by linear-Gaussian dynamics.

Particle Filter

- *Particle filters* represent the belief states as a collection of states.
- Each state in the approximated belief is called a *particle*.
- Useful in problems with large discrete states spaces or continuous problems not well approximated by linear-Gaussian dynamics.

Algorithm 1 Particle filter algorithm.

function ParticleFilter($\mathbf{s}, T, O, a, o$)

Particle Filter

- *Particle filters* represent the belief states as a collection of states.
- Each state in the approximated belief is called a *particle*.
- Useful in problems with large discrete states spaces or continuous problems not well approximated by linear-Gaussian dynamics.

Algorithm 1 Particle filter algorithm.

---

function ParticleFilter($\mathbf{s}, T, O, a, o$)
  $\mathbf{s}' \sim T(\mathbf{s}, a)$ ▷ next states

Particle Filter

- *Particle filters* represent the belief states as a collection of states.
- Each state in the approximated belief is called a *particle*.
- Useful in problems with large discrete states spaces or continuous problems not well approximated by linear-Gaussian dynamics.

Algorithm 1 Particle filter algorithm.

function ParticleFilter($\mathbf{s}, T, O, a, o$)
  $\mathbf{s}' \sim T(\mathbf{s}, a)$ ▷ next states
  $\mathbf{w} \leftarrow O(o \mid \mathbf{s}', a)$ ▷ weights

Particle Filter

- *Particle filters* represent the belief states as a collection of states.
- Each state in the approximated belief is called a *particle*.
- Useful in problems with large discrete states spaces or continuous problems not well approximated by linear-Gaussian dynamics.

Algorithm 1 Particle filter algorithm.

---

function PARTICLEFILTER($\mathbf{s}, T, O, a, o$)

$\mathbf{s}' \sim T(\mathbf{s}, a)$ ▷ next states

$\mathbf{w} \leftarrow O(o \mid \mathbf{s}', a)$ ▷ weights

particles $\sim \text{SetCategorical}\left(\mathbf{s}', \frac{\mathbf{w}}{\sum_i w_i}\right)$ ▷ sample with normalized weights

Particle Filter

- *Particle filters* represent the belief states as a collection of states.
- Each state in the approximated belief is called a *particle*.
- Useful in problems with large discrete states spaces or continuous problems not well approximated by linear-Gaussian dynamics.

Algorithm 1 Particle filter algorithm.

function ParticleFilter($\mathbf{s}, T, O, a, o$)
  $\mathbf{s}' \sim T(\mathbf{s}, a)$ ▷ next states
  $\mathbf{w} \leftarrow O(o \mid \mathbf{s}', a)$ ▷ weights
  particles $\sim$ SetCategorical $\left(\mathbf{s}', \frac{\mathbf{w}}{\sum_i w_i}\right)$ ▷ sample with normalized weights
  return particles

Particle Filter

- *Particle filters* represent the belief states as a collection of states.
- Each state in the approximated belief is called a *particle*.
- Useful in problems with large discrete states spaces or continuous problems not well approximated by linear-Gaussian dynamics.

Algorithm 1 Particle filter algorithm.

$$
\begin{array}{ll}
\textbf{function } \textsc{ParticleFilter}(\mathbf{s}, T, O, a, o) & \\
\quad \mathbf{s}' \sim T(\mathbf{s}, a) & \triangleright \text{ next states} \\
\quad \mathbf{w} \leftarrow O(o \mid \mathbf{s}', a) & \triangleright \text{ weights} \\
\quad \text{particles} \sim \text{SetCategorical}\left(\mathbf{s}', \frac{\mathbf{w}}{\sum_i w_i}\right) & \triangleright \text{ sample with normalized weights} \\
\quad \textbf{return } \text{particles} &
\end{array}
$$

See Pluto notebook: StateEstimation.jl/particle_filter.jl

Particle Filter Variants

Particle filter with rejection:

- Used in problems with discrete observations.
- Any sampled observation that does not equal the true observation is rejected.
- Problem of *particle deprivation*: lack of particles near the true state.[11]

Injection particle filter:

- Inject random particles to protect against particle deprivation.

Adaptive injection particle filter:

- Inject particles adaptively based on a ratio of two exponentially moving averages of the mean particle weights (using *fast* and *slow* moving averages).

[11] Due to low particle coverage given the stochastic nature of resampling.

Kalman Filter

To update beliefs with *continuous* state spaces, we integrate instead of sum:

$$b'(s') \propto O(o \mid a, s') \int T(s' \mid s, a) b(s) \, ds$$

A *Kalman filter* assumes that T and O are linear-Gaussian and b is Gaussian:

$$\begin{aligned} T(\mathbf{s}' \mid \mathbf{s}, \mathbf{a}) &= \mathcal{N}(\mathbf{s}' \mid \mathbf{T}_s \mathbf{s} + \mathbf{T}_a \mathbf{a}, \, \mathbf{\Sigma}_s) \\ O(\mathbf{o} \mid \mathbf{s}') &= \mathcal{N}(\mathbf{o} \mid \mathbf{O}_s \mathbf{s}', \, \mathbf{\Sigma}_o) \\ b(\mathbf{s}) &= \mathcal{N}(\mathbf{s} \mid \boldsymbol{\mu}_b, \, \mathbf{\Sigma}_b) \end{aligned}$$

See Pluto notebook: StateEstimation.jl/kalman_filter.jl

POMDP solvers

A number of ways to solve POMDPs are implemented in the following packages.

Table: `POMDPs.jl` Solution Methods

Package	**Offline/*Online***	**State Spaces**	**Actions Spaces**	**Observation Spaces**
`QMDP.jl`	Offline	Discrete	Discrete	Discrete
`FIB.jl`	Offline	Discrete	Discrete	Discrete
`BeliefGridValueIteration.jl`	Offline	Discrete	Discrete	Discrete
`SARSOP.jl`	Offline	Discrete	Discrete	Discrete
`POMDPSolve.jl`	Offline	Discrete	Discrete	Discrete
`IncrementalPruning.jl`	Offline	Discrete	Discrete	Discrete
`PointBasedValueIteration.jl`	Offline	Discrete	Discrete	Discrete
`MCVI.jl`	Offline	Continuous	Discrete	Continuous
`AEMS.jl`	*Online*	Discrete	Discrete	Discrete
`BasicPOMCP.jl`	*Online*	Continuous	Discrete	Discrete
`ARDESPOT.jl`	*Online*	Continuous	Discrete	Discrete
`AdaOPS.jl`	*Online*	Continuous	Discrete	Continuous
`POMCPOW.jl`	*Online*	Continuous	Continuous	Continuous
`BetaZero.jl`	Offline + *Online*	Continuous	Discrete	Continuous

When defining your problem, the ***type*** of state, action, and observation space is very important!

Questions

Any questions?

(feel free to post on Ed or email me: mossr@cs.stanford.edu)

Slides and code: `https://github.com/sisl/StateEstimation.jl`

References

Kochenderfer, Mykel J., Tim A. Wheeler, and Kyle H. Wray. *Algorithms for Decision Making*. MIT Press, 2022.