Skoltech

Skolkovo Institute of Science and Technology

L08: Planning under Uncertainty

Planning Algorithms in AI

Prof. Gonzalo Ferrer,

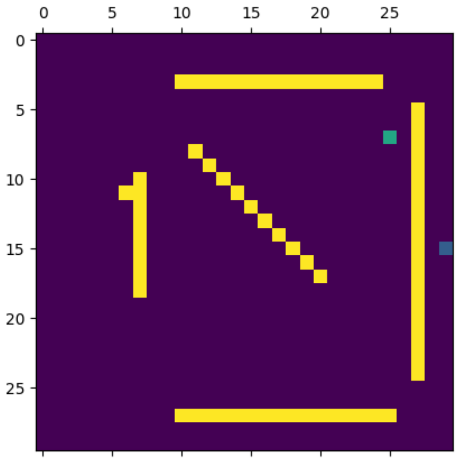
Skoltech, 5 December 2022

Summary L08, Sutton and Barto, Ch3 plus papers

- Uncertainty in State Propagation
- The Markov property
- Reward and Return
- Markov Decision Process (MDP)
- Stochastic Policy
- Bellman equation
- Partially observable MDP
- Belief Space Planning

Uncertainty in State Propagation

- From last lecture, we presented nature as a mysterious and unpredictable source of uncertainty for the robot (our agent).
- Now we can consider that a real system. There might be a difference between what we wanted to do (planned action) and what it really happened (executed action) after propagating a state.



Maze pursuer choosing next action (PS3).

Wheeled robot Akula traversing uneven terrain.

Probabilistic Transition between States

- Transitioning between states considering uncertainty, two options
    - Uncertainty in the action space: $s' = f(s, a + \epsilon), \quad \text{where } \epsilon \text{ is a r.v.}$
    - Uncertainty in the state space: $s' = f(s, a) + \eta, \quad \text{where } \eta \text{ is a r.v.}$
- Note we have changed the convention to indicate states *s* and actions *a*.
- Since there is a random variable involved, we can build the transition probability from any state *s* and any action *a*:

$$p(s'|s,a) = Pr\{S' = s'|S = s, A = a\}$$

Sequences of State Propagation

- A sequence plan is executed by following a sequence of actions (policy) and propagating states over time. For the deterministic case:

$$s_{t+1} = f(s_t, a_t)$$

- Now, for the propagation of states, we need to determine all r.v. affecting the propagation:

$$s_{t+1} = f(s_t, a_t) = f(f(s_{t-1}, a_{t-1}), a_t)) = f(f(f(s_{t-2}, a_{t-2}), a_{t-1}), a_t))$$

- It results in a recursive form where the probability of transition is:

$$p(s_{t+1} | s_1, a_1, \ldots, s_t, a_t)$$

The Markov property

The future only depends on the present and we can discard the past.

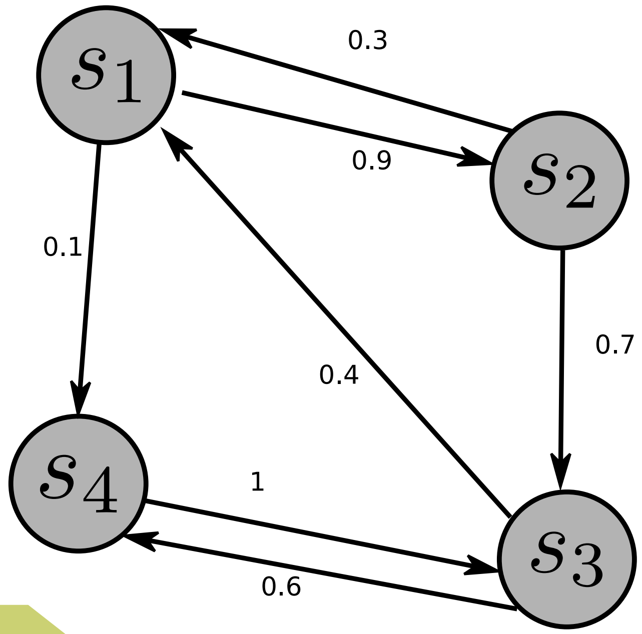
$$p(s_{t+1}|s_1, \dots, s_t) = p(s_{t+1}|s_t)$$

- The current state at time *t* captures all relevant information for prediction.
- There is no need to store past states if the states are Markov.
- The same idea can be applied for probability transitions:

$$p(s_{t+1}|s_1, a_1, \dots, s_t, a_t) = p(s_{t+1}|s_t, a_t)$$

Markov Process (Markov Chain)

- The most straightforward way to use the Markov property is in the form or a Markov Process or Markov Chain.
- The elements are a state space S and a transition Probability matrix P between states.



$p(s' \mid s)$

to

from	s_1'	s_2	s_3	s_4
s_1	0	0.9	0	0.1
s_2	0.3	0	0.7	0
s_3	0.4	0	0	0.6
s_4	0	0	1	0

Reward

- At each time step, the reward is a simple number $R_t \in \mathbb{R}$
- The agent's goal is to maximize the total amount of reward it receives maximizing not immediate reward, but cumulative reward in the long run.
- The *reward hypothesis*:

  "That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward)."

Example:

- A robot getting to its goal may receive +1 or -1 for a collision.

- Students get grades after submitting PS's

Return

- The **return** is cumulative reward summing for all the rewards until the end of the episode, at the terminal state time index T:

$$G_t = R_{t+1} + R_{t+2} + \ldots + R_T$$

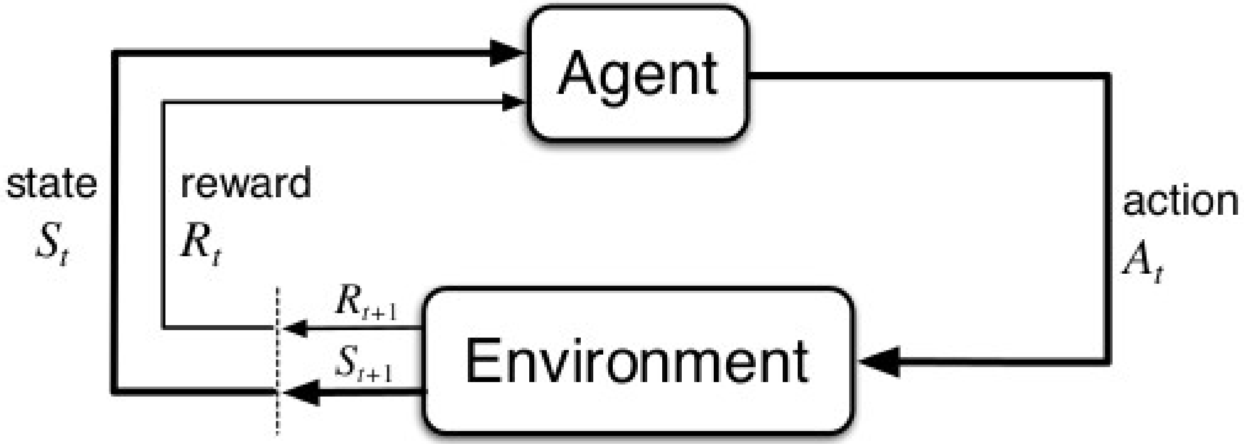
- This presents a problem for *continuous tasks*, those that never end.
- Then, one can define a **discount rate** $\gamma \in [0, 1]$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- We could unite both returns if we consider a **terminal action** (or an *absorbing state*) whose reward is zero. This way, finite plans can be solved and their solution is unmodified even with discount.

Markov Decision Process

Finally, we have all components to define a Markov Decision Process. In essence, an MPD is an *environment* with the following components:



This scheme we use for planning, but in general is also valid for most of the theory behind reinforcement learning.

Markov Decision Process

Finally, we have all components to define a Markov Decision Process.
In essence, an MPD is an *environment* with the following components:

More formally, MDP is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- Finite set of states $\mathcal{S}$
- Finite set of actions $\mathcal{A}$
- State transition probability matrix

$$\mathcal{P}^a_{s's} = Pr\{S_{t+1} = s' | S_t = s, A_t = a\}$$

- Reward function $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- Discount factor $\gamma \in [0, 1]$

Markov Decision Process

After defining all components, we can do a simplification in notation by considering the joint probability

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$$

Now, we can express the all the required elements of the MDP in terms of only this probability, which will simplify notation. For example:

- Expected reward of action-state: $r(s, a) = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a)$
- State transition probabilities

$$p(s'|s, a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(r, s', r|s, a)$$

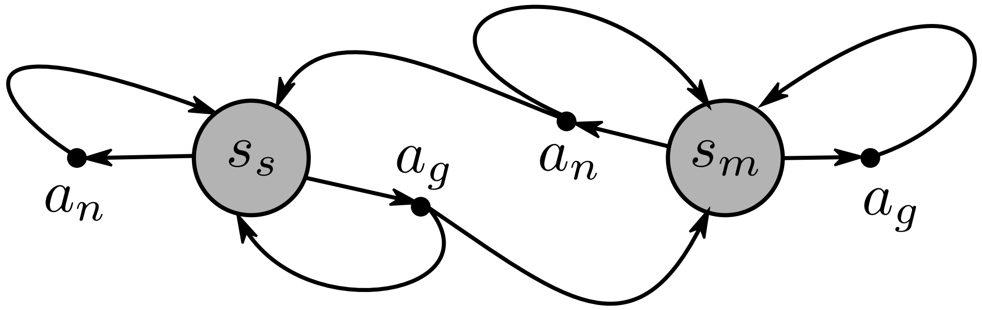
- Expected reward for action-state-nextstate

$$r(s, a, s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r p(s', r|s, a)}{p(s'|s, a)}$$

Example MPD: Transition Graph

$\mathcal{A} = \{a_{\text{nothing}}, a_{\text{gas}}\} = \{a_n, a_g\}$

$\mathcal{S} = \{s_{\text{moving}}, s_{\text{stopped}}\} = \{s_{\text{m}}, s_{\text{s}}\}$



$p(r, s'|a, s)$

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
s_s	s_s	a_n	1	0
s_s	s_s	a_{gas}	0.3	0
s_s	s_m	a_{gas}	0.7	r_{moving} !
s_s	s_m	a_n	0	$r_{mov.}$
s_m	s_m	a_n	0.9	$r_{mov.}$
s_m	s_s	a_n	0.1	$-r_{stopped}$

Stochastic Policies: Action Selection

- Previously in this course we have studied action selection as a **deterministic policy**, where the plan states which is the next (best) action to execute:

$$\pi(s) = a$$

- Now, one can model the effect of uncertainty as a **stochastic policy**, a probability (for discrete spaces), that reflects the randomness brought by natural effects, imperfections in modeling, etc.

$$\pi(a|s) = Pr\{A_t = a|S_t = s\}, \quad \text{where } s \in \mathcal{S}, a \in \mathcal{A}(s)$$

- As a result of the MDP, we will consider stochastic policies to be Markovian:

$$\pi(a_t|s_1, \ldots, s_t) = \pi(a_t|s_t)$$

Value Function

- The **state-value function** for policy *pi* is:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right]$$

This *value* gives an estimate of "how good" given a state. We have seen examples of this in the accumulated cost *L()* in lecture 5.

The only novelty here is that the policy is of stochastic nature, but still one can solve this by using the Expectation operator.

- Similarly, we can define the **action-value function** for policy *pi*:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]$$

Bellman equation

- From the *value-state function* a recursive form arises, the ***Bellman equation for v$_{\pi}$***:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big],
\end{aligned}
$$

There are several ways of solving this. We already studied one of them *Dynamic programming* which requires a complete backup of all states.
A different alternative is solving directly the linear system

Optimal Value Functions

- The previous example solved the Bellman equation for a policy, but we never said it was a "good" policy.
- The the notion of optimality defined in the ***optimal state-value function***:

$$v_*(s) = \max_\pi v_\pi(s), \forall s \in \mathcal{S}$$

  Here we are comparing all possible policies and an optimal policy is the best

- Similarly, the ***optimal action-value function*** is defined as:

$$q_*(s,a) = \max_\pi q_\pi(s,a), \quad \forall s \in \mathcal{S} \text{ and } \forall a \in \mathcal{A}(s)$$

  This function gives the expected return for taking action a in state s thereafter following an optimal policy.

- We can also write *q* in terms of *v*: $q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$

Optimal Value Functions

A recursive form arises, the ***Bellman optimality equation***:

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi^*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big].
\end{aligned}
$$

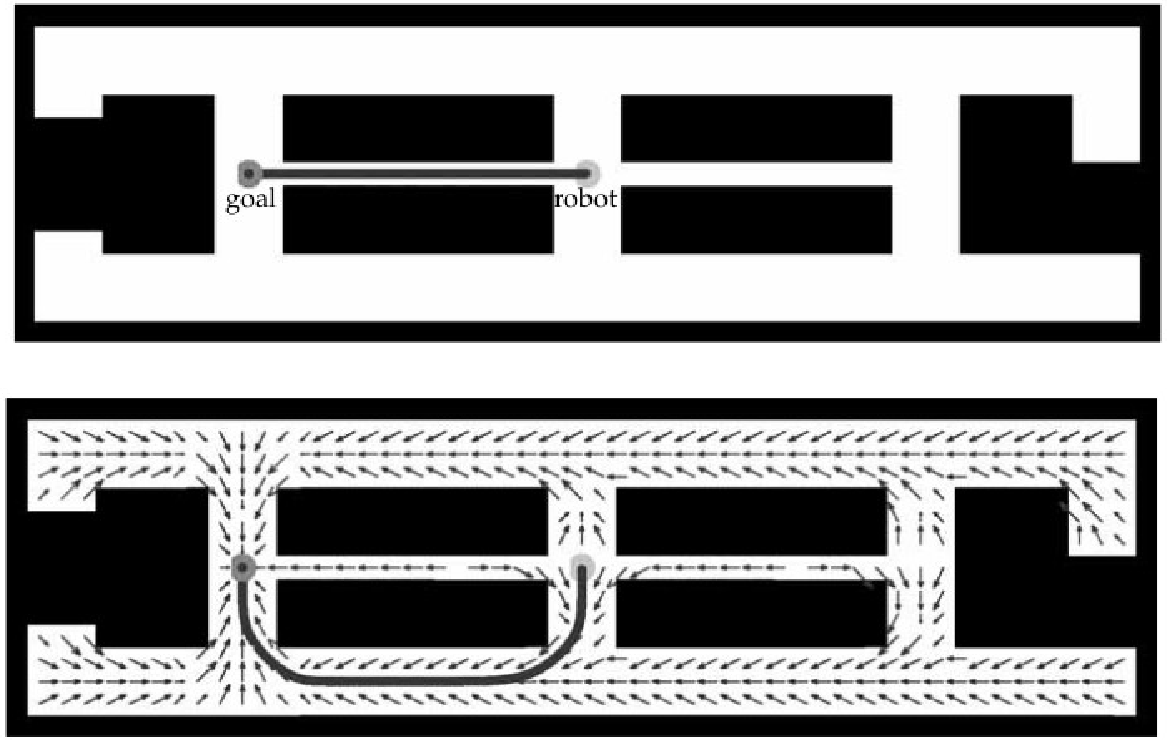
Optimal Value Functions

- The Bellman optimality equation: the value of a state under an optimal policy must equal the expected return for the best action from that state.
- Once the Optimal Bellman Equation has been solved, there is a simple way to calculate an optimal policy: a *greedy* approach just following the best (expected) value.

$$a^*(s) = \max_a q_*(s, a)$$

- We need to ensure that the probability of transition is not zero, but other than that, this short-term search would provide the long-term optimal path.

MDP: Example of robot navigation



The shortest path is not the solution for the following navigation problem, where the reward penalizes collisions.

MDP, remarks

- (1) we accurately know the dynamics of the environment;
- (2) we have enough computational resources to complete the computation of the solution (tabular cases so far)
- (3) the Markov property.
- In practice, direct MDP or any other discretization is not easy to implement and we will approximate solutions to MDP.
- Some (even more complicated) problems can be approximated as MDP.

Partially Observable MDP

- In addition to uncertainty in action imagine now there is uncertainty in the state, and it is not possible to know for certain what it is:

$$p(z|s)$$

- Now the state *s* is not directly observable and one simply can gather observations *z* from sensors for example.
- This is known as Partially Observable Markov Decision Process (POMDP)
- The motivation is that planning now becomes a process were not just a return is optimized, the algorithm anticipates *certainty* , actively gather information and explore optimality while planning.
- In brief, if MDP had a problem of feasibility for tabular problems, things get much worse here, to the point that only small grid worlds allow to solve POMDP exhaustively.

Belief Space or Information Space

- Same as the C-space modeled the robot/agents state variables, we can also take into account these “extra” dimensions due to observations.
- In the **belief space**, we would include additional variables to the robot configuration capturing uncertainty.
- How to represent this? This is a research question.
- How to propagate states? It also depends on the approach.
- In brief, the topic of **Belief Space Planning**.