16.410/413
Principles of Autonomy and Decision Making
Lecture 22: Markov Decision Processes I

Emilio Frazzoli

Aeronautics and Astronautics
Massachusetts Institute of Technology

November 29, 2010

Assignments

Readings

- Lecture notes
- [AIMA] Ch. 17.1-3.

Outline

From deterministic to stochastic planning problems

A basic planning model for deterministic systems (e.g., graph/tree search algorithms, etc.) is :

Planning Model (Transition system + goal)

A (discrete, deterministic) feasible planning model is defined by

- A countable set of states $\mathcal{S}$.
- A countable set of actions $\mathcal{A}$.
- A transition relation $\rightarrow \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$.
- An initial state $s_1 \in \mathcal{S}$.
- A set of goal states $s_{\mathrm{G}} \subset \mathcal{S}$.

- We considered the case in which the transition relation is purely deterministic: if (s, a, s') are in relation, i.e., $(s, a, s') \in \rightarrow$, or, more concisely, $s \xrightarrow{a} s'$, then taking action a from state s will always take the state to s'.
- Can we extend this model to include (probabilistic) uncertainty in the transitions?

Markov Decision Process

Instead of a (deterministic) transition relation, let us define transition probabilities; also, let us introduce a reward (or cost) structure:

Markov Decision Process (Stoch. transition system + reward)

A Markov Decision Process (MDP) is defined by

- A countable set of states $\mathcal{S}$.
- A countable set of actions $\mathcal{A}$.
- A transition probability function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$.
- An initial state $s_0 \in \mathcal{S}$.
- A reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$.

- In other words: if action a is applied from state s, a transition to state s' will occur with probability $T(s, a, s')$.
- Furthermore, every time a transition is made from s to s' using action a, a reward $R(s, a, s')$ is collected.

Some remarks

- In a Markov Decision Process, both transition probabilities and rewards only depend on the present state, not on the history of the state. In other words, the future states and rewards are independent of the past, given the present.
- A Markov Decision Process has many common features with Markov Chains and Transition Systems.
- In a MDP:
  - Transitions and rewards are stationary.
  - The state is known exactly. (Only transitions are stochastic.)
- MDPs in which the state is not known exactly (HMM + Transition Systems) are called Partially Observable Markov Decision Processes (POMDP's): these are very hard problems.

Total reward in a MDP

- Let us assume that it is desired to maximize the total reward collected over infinite time.
- In other words, let us assume that the sequence of states is $S = (s_1, s_2, \ldots, s_t, \ldots)$, and the sequence of actions is $A = (a_1, a_2, \ldots, a_t, \ldots)$; then the total collected reward (also called utility) is

$$V = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}),$$

  where $\gamma \in (0, 1]$ is a discount factor.
  - Philosophically: it models the fact that an immediate reward is better than an uncertain reward in the future.
  - Mathematically: it ensures that the sum is always finite, if the rewards are bounded (e.g., finitely many states/actions).

Decision making in MDPs

- Notice that the actual sequence of states, and hence the actual total reward, is unknown a priori.
- We could choose a plan, i.e., a sequence of actions: $A = (a_1, a_2, \ldots)$.
- In this case, transition probabilities are fixed and one can compute the probability of being at any given state at each time step—in a similar way as the forward algorithm in HMMs—and hence compute the expected reward:

$$\mathrm{E}[R(s_t, a_t, s_{t+1}) | s_t, a_t] = \sum_{s \in \mathcal{S}} T(s_t, a_t, s) R(s_t, a_t, s)$$

- Such approach is essentially open loop, i.e., it does not take advantage of the fact that at each time step the actual state reached is known, and a new feedback strategy can be computed based on this knowledge.

Introduction to value iteration

- Let us assume we have a function $V_i : \mathcal{S} \to \mathcal{R}_+$ that associates to each state s a lower bound on the optimal (discounted) total reward $V^*(s)$ that can be collected starting from that state. *Note the connection with admissible heuristics in informed search algorithms.*
- For example, we can start with $V_0(s) = 0$, for all $s \in \mathcal{S}$.
- As a feedback strategy, we can do the following: at each state, choose the action that maximizes the expected reward of the present action + estimate total reward from the next step onwards.
- Using this strategy, we can get an update V_{i+1} on the function V_i.
- Iterate until convergence...

Value iteration algorithm

A bit more formally:

- Set $V_0(s) \leftarrow 0$, for all $s \in \mathcal{S}$
- iterate, for all $s \in \mathcal{S}$:

$$
\begin{aligned}
V_{i+1}(s) &\leftarrow \max_a \mathrm{E}\left[R(s,a,s') + \gamma V_i(s')\right] \\
&= \max_a \sum_{s' \in \mathcal{S}} T(s,a,s')\left[R(s,a,s') + \gamma V_i(s')\right],
\end{aligned}
$$

until $\max_s |V_{i+1}(s) - V_i(s)| < \epsilon$.

Value iteration example

Let us consider a simple MDP:

- The state space is a 10-by-10 grid.
- The border cells and some of the interior cells are "obstacles" (marked in gray).
- The initial state is the top-left feasible cell.
- A reward of 1 is collected when reaching the bottom right feasible cell. The discount factor is 0.9.
- At each non-obstacle cell, the agent can attempt to move to any of the neighboring cells. The move will be successful with probability 3/4. Otherwise the agent will move to a different neighboring cell, with equal probability.
- The agent always has the option to stay put, which will succeed with certainty.

Value iteration example

Initial condition:

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Value iteration example

After 1 iteration:

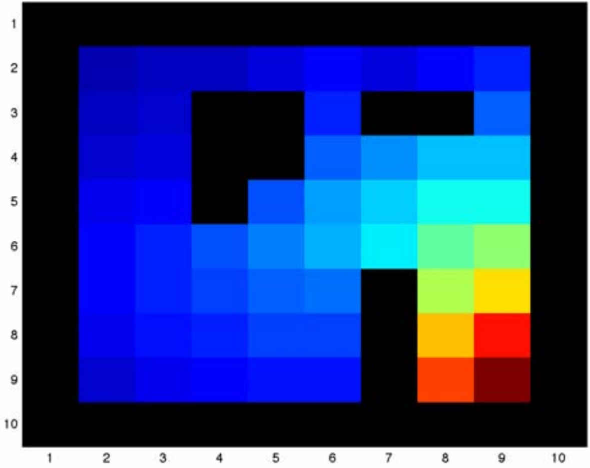
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.75	0
0	0	0	0	0	0	0	0.75	1	0
0	0	0	0	0	0	0	0	0	0

Value iteration example

After 2 iterations:

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0.51	0
0	0	0	0	0	0	0	0.56	1.43	0
0	0	0	0	0	0	0	1.43	1.9	0
0	0	0	0	0	0	0	0	0	0

Value iteration example



Value iteration example

After 50 iterations:

0	0	0	0	0	0	0	0	0	0
0	0.44	0.54	0.59	0.82	1.15	0.85	1.09	1.52	0
0	0.59	0.69	0	0	1.52	0	0	2.13	0
0	0.75	0.90	0	0	2.12	2.55	2.98	3.00	0
0	0.95	1.18	0	2.00	2.70	3.22	3.80	3.88	0
0	1.20	1.55	1.87	2.41	2.92	3.51	4.52	5.00	0
0	1.15	1.47	1.74	2.05	2.25	0	5.34	6.47	0
0	0.99	1.26	1.49	1.72	1.74	0	6.69	8.44	0
0	0.74	0.99	1.17	1.34	1.27	0	7.96	9.94	0
0	0	0	0	0	0	0	0	0	0

Bellman's equation

- Under some technical conditions (e.g., finite state and action spaces, and $\gamma < 1$), value iteration converges to the optimal value function V^*.
- The optimal value function V^* satisfies the following equation, called the Bellman's equation, a nice (perhaps the prime) example of the principle of optimality

$$\begin{aligned} V^*(s) &= \max_a \mathrm{E}\left[R(s,a,s') + \gamma V^*(s')\right] \\ &= \max_a \sum_{s' \in \mathcal{S}} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right], \qquad \forall s \in \mathcal{S}. \end{aligned}$$

- In other words, the optimal value function can be seen as a fixed point for value iteration.
- The Bellman's equation can be proven by contradiction.

Value iteration summary

- Value iteration converges monotonically and in polynomial time to the optimal value function.
- The optimal policy can be easily recovered from the optimal value function:

$$\pi^*(s) = \arg\max_a \mathrm{E}\left[R(s, a, s') + \gamma V^*(s')\right], \qquad \forall s \in \mathcal{S}.$$

- Knowledge of the value function turns the optimal planning problem into a feedback problem,
    - Robust to uncertainty
    - Minimal on-line computations