16.410/413
Principles of Autonomy and Decision Making

Lecture 23: Markov Decision Processes
Policy Iteration

Emilio Frazzoli

Aeronautics and Astronautics
Massachusetts Institute of Technology

December 1, 2010

Assignments

Readings

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

Searching over policies

- Value iteration converges exponentially fast, but still asymptotically.
- Recall how the best policy is recovered from the current estimate of the value function:

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

- In order to figure out the optimal policy, it should not be necessary to compute the optimal value function exactly...
- Since there are only finitely many policies in a finite-state, finite-action MDP, it is reasonable to expect that a search over policies should terminate in a finite number of steps.

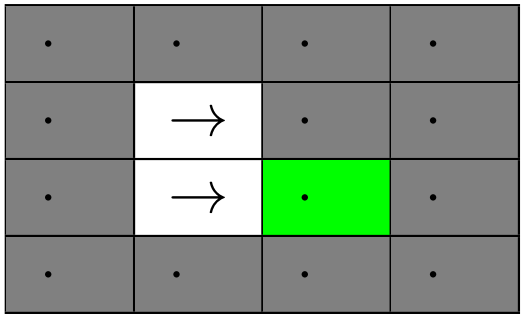
Policy evaluation

- Let us assume we have a policy, e.g., $\pi : \mathcal{S} \to \mathcal{A}$, that assigns an action to each state. I.e., action $\pi(s)$ will be chosen each time the system is at state s.
- Once the actions taken at each state are fixed,
  - the MDP is turned into a Markov chain (with rewards).
  - one can compute the expected utility collected over time using that policy
- In other words, one can evaluate how well a certain policy does by computing the value function induced by that policy.

Policy evaluation example — naïve method

- Same planning problem as the previous lecture, in a smaller world (4x4).
- Simple policy π: always go right, unless at the goal (or inside obstacles).
- Expected utility (value function) starting from top left corner (cell $2,2$):

$$V_\pi(2,2) \approx 0.06 \cdot 8.1 = 0.5$$



Path	Prob.	Utility
$\rightarrow$	0.75	0
$\uparrow$	0.08	0
$\leftarrow$	0.08	0
$\downarrow\rightarrow$	0.06	8.1
$\ldots$	$\ldots$	$\ldots$

Policy evaluation

- Recalling the MDP properties, one can write the value function at a state as the expected reward collected at the first step + expected discounted value at the next state under the given policy

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

- Note that this is a set of $\mathrm{card}(\mathcal{S})$ linear equations in the $\mathrm{card}(\mathcal{S})$ unknowns $\{V_\pi(s), s \in \mathcal{S}\}$.

- This can be solved efficiently, in $O(\mathrm{card}(\mathcal{S})^3)$

Policy evaluation example

- Let us consider the variables $v_{2,2}, v_{3,2}, v_{3,3}$ (the others are trivially 0).
- We have:

$$\begin{aligned} v_{2,2} &= \frac{3}{4}(0 + 0.9 \cdot 0) + \frac{1}{12}(0 + 0.9 v_{3,2}) \\ v_{3,2} &= \frac{3}{4}(1 + 0.9 v_{3,3}) + \frac{1}{12}(0 + 0.9 v_{2,2}) \\ v_{3,3} &= 1(1 + 0.9 v_{3,3}) \end{aligned}$$

- Solving, we get:

$$\begin{aligned} v_{3,3} &= 10 \\ v_{2,2} &= \frac{3}{40} v_{3,2} = \ldots = 0.5657 \\ v_{3,2} &= 7.5 + \frac{9}{1600} v_{3,2} = \frac{1600}{1591}\, 7.5 = 7.5424 \end{aligned}$$

Policy π

·	·	·	·
·	→	·	·
·	→	·	·
·	·	·	·

Value function V_π

0	0	0	0
0	0.566	0	0
0	7.542	10	0
0	0	0	0

Roll-out policies

- Given a baseline policy π_0, with induced value function V_{π_0}, we can always get another policy π_1 that is at least as good (i.e., such that $V_{\pi_1}(s) \geq V_{\pi_0}(s)$, for all $s \in \mathcal{S}$.

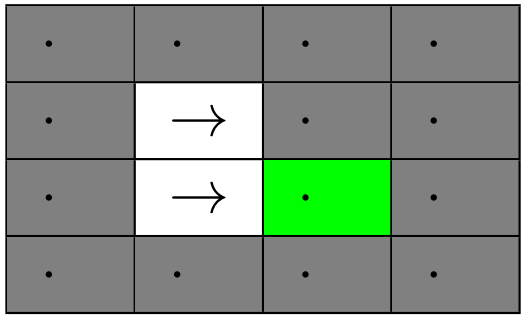
- Idea: for each state s, choose the action that maximizes the expected total reward that will be collected if the baseline policy is used from the next step onwards, i.e.,

$$\begin{aligned}\pi_1(s) &= \arg\max_{a \in \mathcal{A}} \mathrm{E}\left[R(s,a,s') + \gamma V(s')\right] \\ &= \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} T(s,a,s')\left[R(s,\pi(s),s') + \gamma V_{\pi_0}(s')\right], \qquad \forall s \in \mathcal{S}\end{aligned}$$

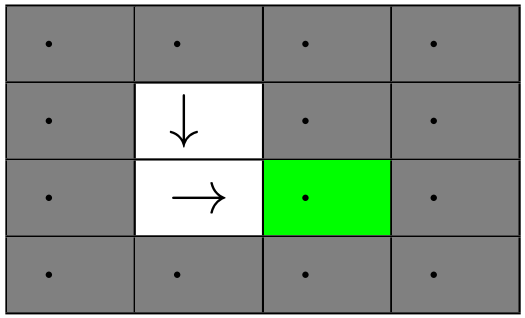
Roll-out policy example

- Baseline policy:
  - $\pi_0(2,2) = \rightarrow$,
  - $\pi_0(3,2) = \rightarrow$,
  - $\pi_0(3,3) = \cdot$.
- $\pi_1(2,2) = \arg\max \begin{cases} \rightarrow: & 0.566 \\ \downarrow: & \mathbf{3/4 \cdot 0.9 \cdot 7.542} \\ \uparrow: & 1/12 \cdot 0.9 \cdot 7.542 \end{cases}$
- $\pi_1(3,2) = \arg\max \begin{cases} \rightarrow: & \mathbf{7.542} \\ \downarrow: & 1/12 \cdot 0.9 \cdot 10 \\ \uparrow: & 1/12 \cdot 0.9 \cdot 0.566 \end{cases}$

Baseline policy π_0



Improved policy π_1



Policy Iteration

- Idea: given a baseline policy, an improved policy can be computed using roll-out. The improved policy can be further improved by applying roll-out again. Repeat.

- Since there are a finite number of states and a finite number of actions, this will eventually terminate with a policy that cannot be further improved

- This is in fact an optimal policy.

Policy Iteration

Policy iteration algorithm:

- Pick an arbitrary policy π.
- iterate:
  1. Policy evaluation: solve the linear system

$$V(s) = \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V(s') \right], \forall s \in \mathcal{S}$$

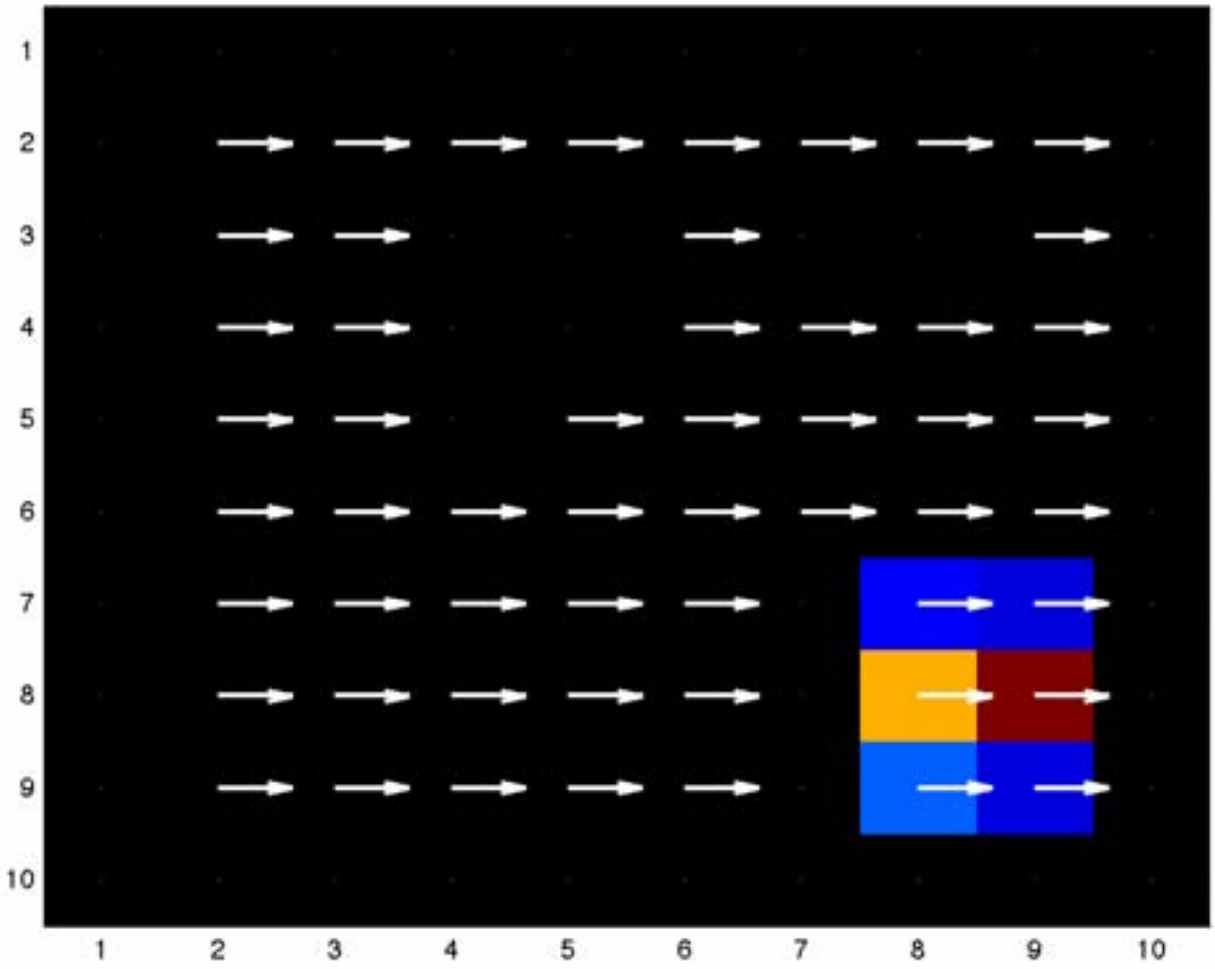
  2. Policy improvement: for each $s \in \mathcal{S}$:

$$\pi(s) \leftarrow \arg\max_a \sum_{s' \in \mathcal{S}} T(s, a, s') \left[R(s, a, s') + \gamma V(s') \right]$$

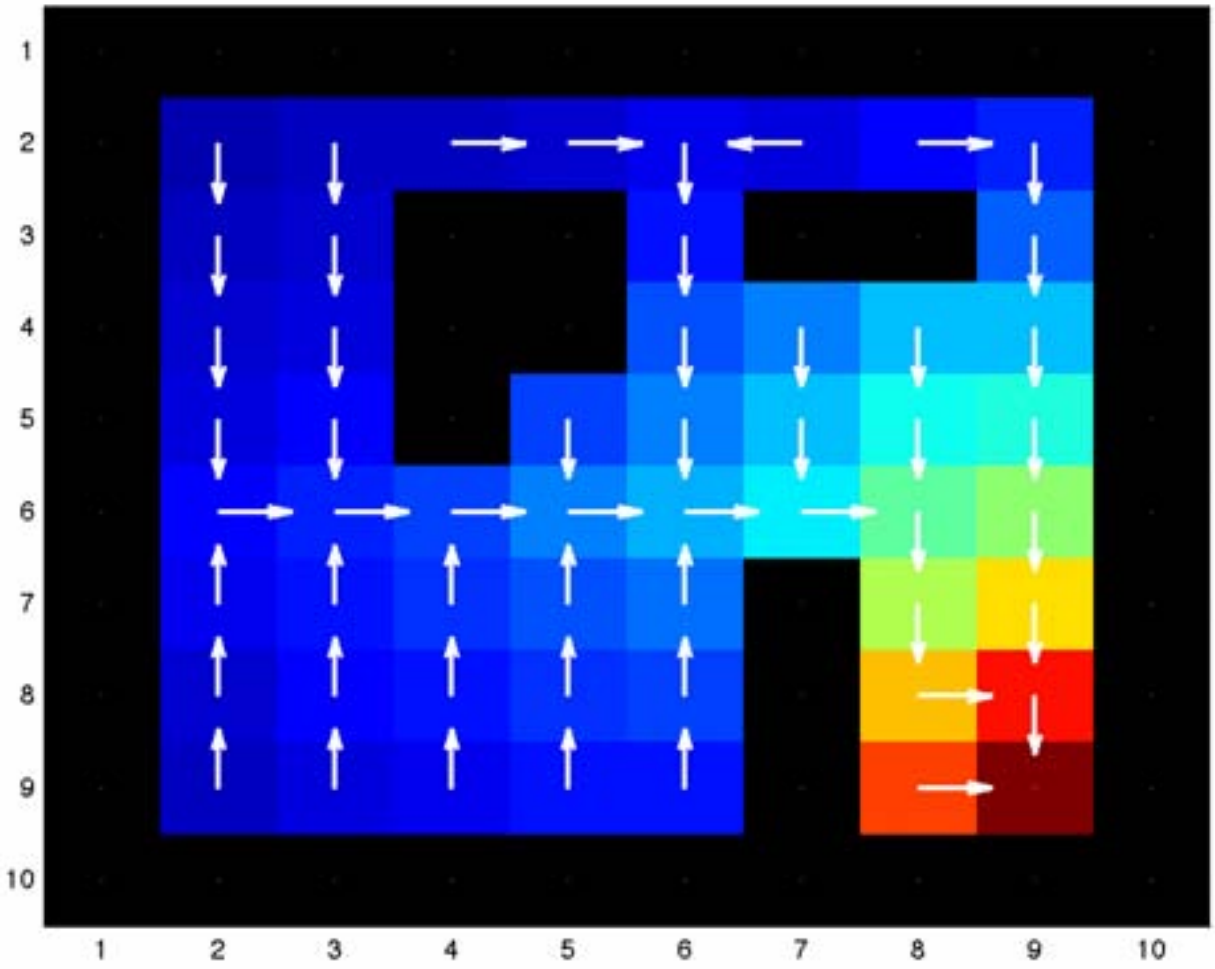
  until π is unchanged.

Policy Iteration example 1/5

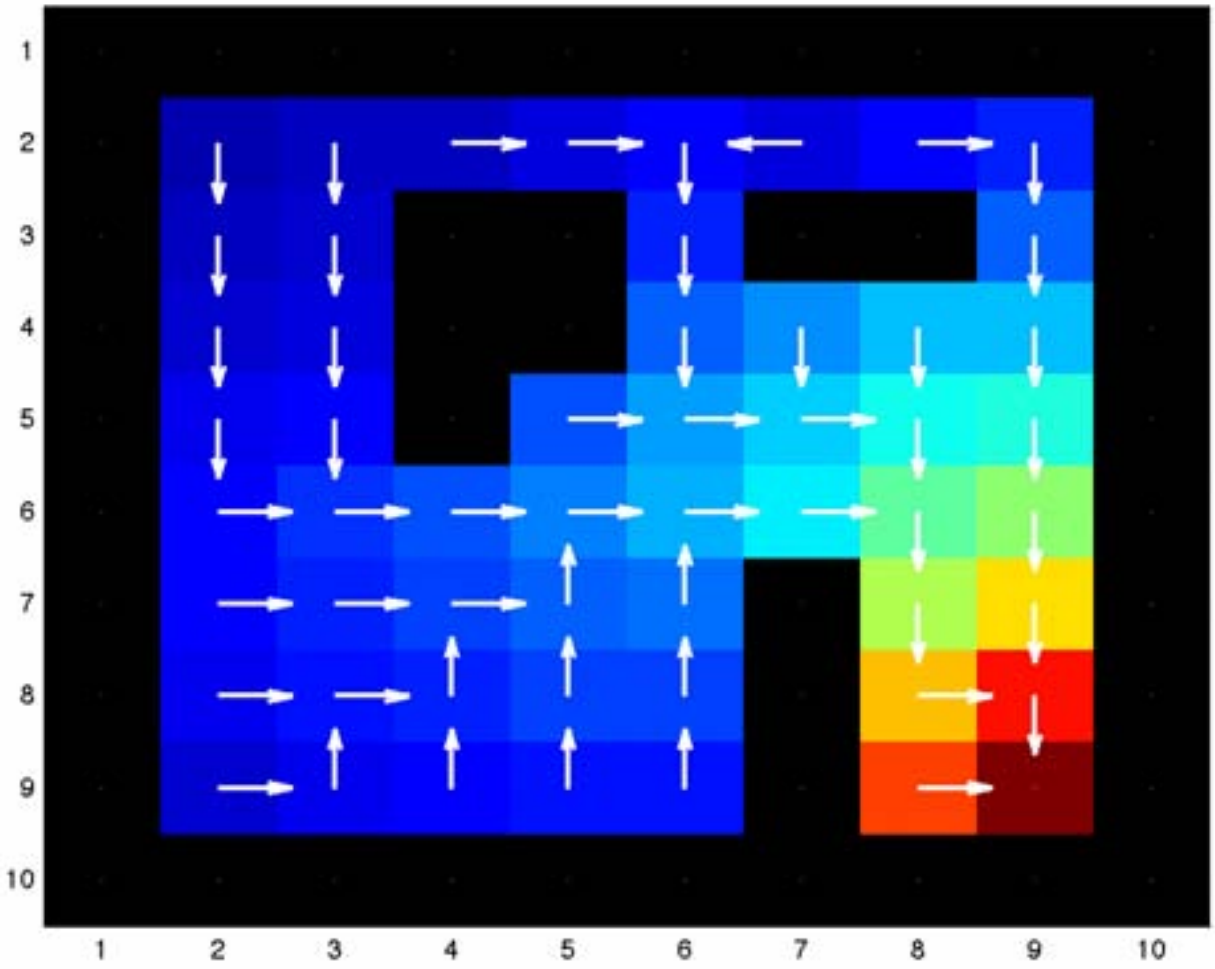
Back to the 10x10 grid:



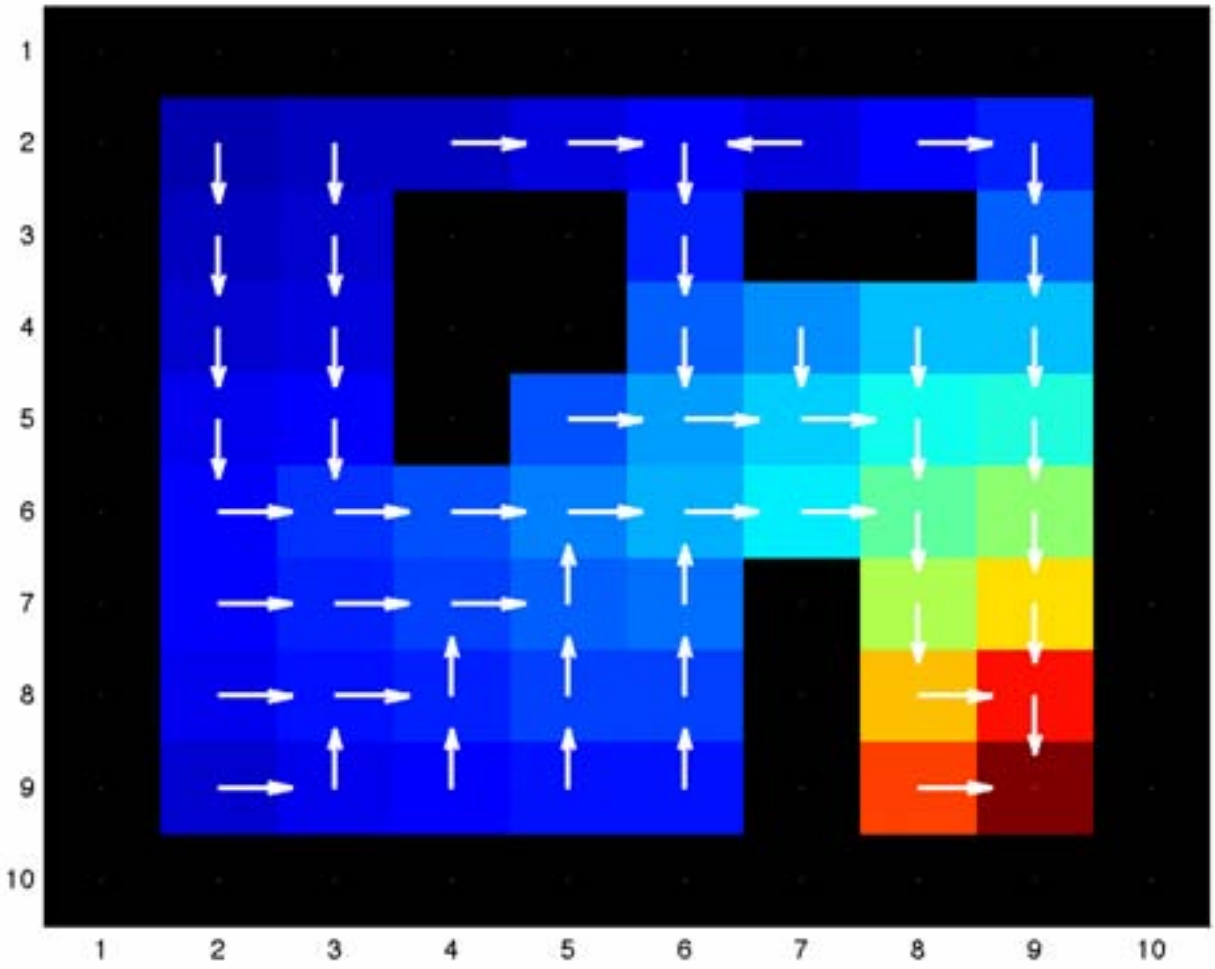
Back to the 10x10 grid:



Back to the 10x10 grid:



Back to the 10x10 grid:



Policy Iteration example 5/5

After 4 iterations:

0	0	0	0	0	0	0	0	0	0
0	0.45	0.56	0.61	0.84	1.17	0.87	1.11	1.5411	0
0	0.61	0.71	0	0	1.54	0	0	2.16	0
0	0.78	0.93	0	0	2.16	2.59	3.02	3.03	0
0	0.98	1.21	0	2.03	2.74	3.26	3.84	3.91	0
0	1.23	1.58	1.90	2.44	2.95	3.54	4.56	5.03	0
0	1.18	1.50	1.78	2.09	2.28	0	5.38	6.51	0
0	1.02	1.29	1.52	1.76	1.77	0	6.74	8.49	0
0	0.76	1.02	1.20	1.37	1.30	0	8.01	10	0
0	0	0	0	0	0	0	0	0	0

This is the optimal value function, induced by the optimal policy (notice exact convergence in a finite number of steps).

Value vs. Policy Iteration

- Policy iteration is desirable because of its finite-time convergence to the optimal policy.
- However, policy iteration requires solving possibly large linear systems: each iteration takes $O(\mathrm{card}(\mathcal{S})^3)$ time.
- Value iteration requires only $O\left(\mathrm{card}(\mathcal{S}) \cdot \mathrm{card}(\mathcal{A})\right)$ time at each iteration — usually the cardinality of the action space is much smaller than that of the state space.

Modified Policy iteration

- Some times, solving the linear system for policy evaluation may be too time consuming (e.g., for large state spaces).

- It turns out that we can get a good approximation of the value function V by doing the following simplified value iteration (simplified since π is given):

$$V_{i+1}(s) = \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V_i(s) \right]$$

Asynchronous Policy Iteration

- In fact, one can even do the following:
- Pick a subset of states $\tilde{\mathcal{S}} \subseteq \mathcal{S}$
- Apply either Value Iteration, or (modified) Policy Iteration to those states.
- Repeat

Model-free MDPs

- In many cases of interest, the exact details of the MDP (i.e., transition probabilities) are not known.
- Reinforcement learning: learn good control policies via the analysis of state/action/rewards sequences collected in simulation and/or experiments.
- Several options, e.g., :
  - Certainty equivalence: estimate transition probabilities through data, then apply standard methods.
    Expensive, not on-line
  - Temporal Difference learning: Only maintain an estimate of the value function V. For each transition, e.g., $s \xrightarrow{a} s'$, update the estimate:

$$V(s) \leftarrow (1-\alpha_t)V(s) + \alpha_t \left[R(s,a,s') + \gamma V(s')\right],$$

    where $\alpha_t \in (0,1)$ is a learning parameter. Note: α_t should decay (e.g., as $\alpha_t = 1/t$) as the number of updated goes to infinity.
    Learning depends on the particular policies applied.

Q-learning

- Estimate total collected reward for state-action pairs.
- Q-factor $Q(s,a)$: estimate of the total collected reward collected (i) starting at state s, (ii) applying action a at the first step, (iii) acting optimally for all future times.
- Q-factor update law, based on an observed transition $s \xrightarrow{a} s'$:

$$Q(s,a) \leftarrow (1-\alpha_t)Q(s,a) + \alpha_t \left[R(s,a,s') + \max_{a'} Q(s',a') \right],$$

  Note: α_t must be decaying over time for convergence, e.g., $\alpha_t = 1/t$.
- Q-learning does not depend on a particular policy.
- Issue: Exploitation (choose "best" a) vs. exploration (choose a poorly characterized a).

Approximation techniques

- Very often (e.g., when the state space $\mathcal{S}$ is a discretization of a continuous state space, e.g., $\mathbb{R}^n$), the dimensions of the state space make value iteration/policy iteration/Q-learning/etc. unfeasible in practice.
- Choose an approximation architecture ϕ, with m parameters r, e.g.,
  - ϕ: basis functions, r: coefficients
  - ϕ: "feature vector", r: coefficients
  - ϕ: neural network, r parameters (weights/biases, etc.)
- Write, e.g.,

$$Q(s,a) = \tilde{Q}(s,a,r) = \sum_{k=1}^{m} r_k \phi_k(s,a)$$

- Updates to $Q(s,a)$ correspond to updates to the (low-dimensional) parameter vector r: find best r such that

$$Q(\cdot,\cdot) \approx \tilde{Q}(\cdot,\cdot,r).$$

16.410 / 16.413 Principles of Autonomy and Decision Making

Fall 2010