Sequential Decision Making

Dynamic programming

Christos Dimitrakakis

Intelligent Autonomous Systems, IvI, University of Amsterdam, The Netherlands

March 18, 2008

The purpose of this 'lecture'

Basic concepts

- Refresh memory.
- Present the MDP setting.
- Define optimality.
- Categorize planning tasks

Algorithms

- Introduce basic planning algorithms.
- Promote intuition about their relationships.
- Discuss their applicability.

Ultimate goal

A firm foundation in reasoning and planning under uncertainty.

Preliminaries

Variables

- Environment $\mu \in \mathcal{M}$
- States $s_t \in \mathcal{S}$.
- Actions $a_t \in \mathcal{A}$.
- A reward $r_t \in \mathbb{R}$.
- A policy $\pi \in \mathcal{P}$.

Notation

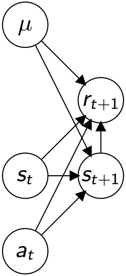
- Probabilities $\mathbf{P}(x|y, z) \equiv z(x|y)$.
- Expectations $\mathbf{E}(x|y, z)$
- Sometimes $\mathbf{P}(a_t = a|\cdot)$ will be used for clarity.
- i.e. $\pi_t(a|s) = \mathbf{P}(a_t = a|s_t = s, \pi_t)$

Markov decision processes

The setting

We are in some dynamic environment μ, where at each time step t we observe

- States $s_t \in \mathcal{S}$.
- Actions $a_t \in \mathcal{A}$.
- A reward $r_t \in \mathbb{R}$.



$$\mathbf{P}(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = \mathbf{P}(s_{t+1} | s_t, a_t, \mu) \tag{1}$$

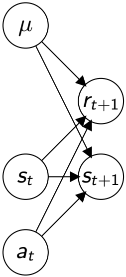
$$p(r_{t+1} | s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = p(r_{t+1} | s_{t+1}, s_t, a_t, \mu) \tag{2}$$

Markov decision processes

The setting

We are in some dynamic environment μ, where at each time step t we observe

- States $s_t \in \mathcal{S}$.
- Actions $a_t \in \mathcal{A}$.
- A reward $r_t \in \mathbb{R}$.



$$\mathbf{P}(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = \mathbf{P}(s_{t+1} | s_t, a_t, \mu) \tag{1}$$

$$p(r_{t+1} | s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = p(r_{t+1} | s_{t+1}, s_t, a_t, \mu) \tag{2}$$

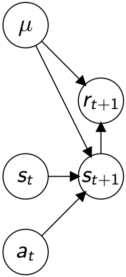
$$p(r_{t+1} | s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \mu) = p(r_{t+1} | s_t, a_t, \mu) \tag{3}$$

Markov decision processes

The setting

We are in some dynamic environment μ, where at each time step t we observe

- States $s_t \in \mathcal{S}$.
- Actions $a_t \in \mathcal{A}$.
- A reward $r_t \in \mathbb{R}$.



$$\mathbf{P}(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots, \mu) = \mathbf{P}(s_{t+1} | s_t, a_t, \mu) \tag{1}$$

$$p(r_{t+1} | s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \dots, \mu) = p(r_{t+1} | s_{t+1}, s_t, a_t, \mu) \tag{2}$$

$$p(r_{t+1} | s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \dots, \mu) = p(r_{t+1} | s_t, a_t, \mu) \tag{3}$$

$$p(r_{t+1} | s_{t+1}, s_t, a_t, s_{t-1}, a_{t-1}, \dots, \mu) = p(r_{t+1} | s_{t+1}, \mu) \tag{4}$$

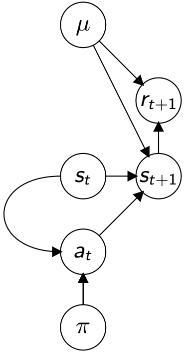
Markov decision processes

Controlling the environment

We wish to control the environment according to some (for now undefined) optimality criterion.

The agent

The agent is fully defined by its policy π. This induces a probability distribution on actions and states.

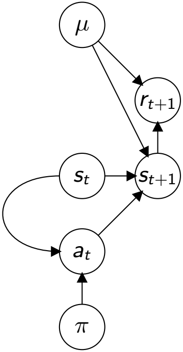


$$\mathbf{P}(a_t | s_t, a_{t-2}, s_{t-1}, a_{t-2}, \ldots, \pi, \mu) = \mathbf{P}(a_t | s_t, \pi) \tag{5}$$

Markov decision processes

The induced Markov chain

Together with the policy π and the model μ, we induce a Markov chain on states.

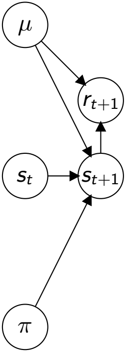


$$\mathbf{P}(s_{t+1}|s_t,\pi,\mu) = \sum_{a\in\mathcal{A}} \mathbf{P}(s_{t+1}|a_t = a, s_t, \pi, \mu)\,\mathbf{P}(a_t = a|s_t,\pi) \tag{6a}$$

$$\mathbf{P}(s_{t+k}|s_t,\pi,\mu) = \sum_{s} \mathbf{P}(s_{t+k}|s_{t+k-1} = s, \pi, \mu)\,\mathbf{P}(s_{t+k-1}|s_t,\pi,\mu) \tag{6b}$$

Note: $\lim_{k\to\infty} \mathbf{P}(s_{t+k} = s|s_t,\pi,\mu)$ is the stationary distribution.

Markov decision processes



The induced Markov chain

Together with the policy π and the model μ, we induce a Markov chain on states.

$$\mathbf{P}(s_{t+1}|s_t, \pi, \mu) = \sum_{a \in \mathcal{A}} \mathbf{P}(s_{t+1}|a_t = a, s_t, \pi, \mu)\, \mathbf{P}(a_t = a|s_t, \pi) \tag{6a}$$

$$\mathbf{P}(s_{t+k}|s_t, \pi, \mu) = \sum_{s} \mathbf{P}(s_{t+k}|s_{t+k-1} = s, \pi, \mu)\, \mathbf{P}(s_{t+k-1}|s_t, \pi, \mu) \tag{6b}$$

Note: $\lim_{k \to \infty} \mathbf{P}(s_{t+k} = s|s_t, \pi, \mu)$ is the stationary distribution.

Planning

The goal in reinforcement learning

To maximise a function of future rewards.

Finite horizon

We are only interested in rewards up to a fixed point in time.

Infinite horizon

We are interested in all rewards.

Value functions

The return / utility

The agent's goal is to maximize the return (Too many Rs, switching to U). For example the utility given a policy π and an MDP μ

$$U^{\pi}_{t,\mu}(\;\cdot\;) \triangleq \mathbf{E}(U|\;\cdot\;, \pi, \mu) = \mathbf{E}\left(\sum_{k=1}^{T} \gamma^k r_{t+k} \middle|\;\cdot\;, \pi, \mu\right) \tag{7}$$

$$= \sum_{k=1}^{T} \gamma^k \sum_{i \in \mathcal{S}} \mathbf{E}[r_{t+k} | s_{t+k}{=}i, \mu]\, \mathbf{P}(s_{t+k} = i|\;\cdot\;, \pi, \mu) \tag{8}$$

Can in principle be calculated from (6).

The value functions

$$V^{\pi}_{t}(s) \triangleq \sum_{a \in \mathcal{A}} U^{\pi}_{t,\mu}(s, a)\pi(a|s) \tag{9}$$

$$Q^{\pi}_{t}(s, a) \triangleq U^{\pi}_{t,\mu}(s, a) \tag{10}$$

Special case: $T \to \infty$, $V^{\pi}_{t}(s) = V^{\pi}(s)$.

Bellman equation

An optimal policy

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

The recursion

$$V_t^\pi(s) = g(t)\,\mathbf{E}[r_{t+1}|s_t{=}s,\pi] + \sum_{k=2}^{T-t} g(t+k)\,\mathbf{E}[r_{t+k}|s_t{=}s, a_t{=}a, \pi, \mu] \tag{11}$$

$$= g(t)\,\mathbf{E}[r_{t+1}|s_t{=}s,\pi] + \sum_{i\in\mathcal{S}} V_{t+1}^\pi(i)\mu(s_{t+1}{=}i|s_t{=}s,\pi). \tag{12}$$

- The current stage's value is just the next reward plus the next stage's value.
- See also the Hamilton-Jacobi-Bellman equation in optimal control.

Greedy policies

The 1-step greedy policy

The 1-step-greedy policy with respect to a given value function can be expressed as

$$\pi(a|s) = \begin{cases} 1, & a = \arg\max_{a'} Q(s, a') \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

The optimal policy

The 1-step-greedy policy with respect to the optimal value function is optimal.

Naive solution

Evaluate all policies, select $\pi^* : V^{\pi^*}(s) \geq V^{\pi}(s) \ \forall s \in \mathcal{S}$.

Clever solutions

- Directly estimate V^*.
- Iteratively improve π.

Problem types

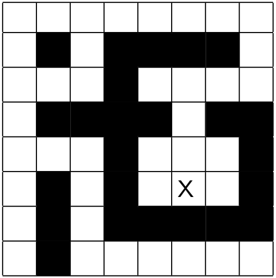
Planning with...

- Finite vs Infinite horizon
- Discounted vs Undiscounted rewards
- Certain vs Uncertain knowledge
- Expected vs worst-case utility functions

Environments

- Deterministic $\leftrightarrow$ Stochastic
- Episodic $\leftrightarrow$ Continuing
- Observable $\leftrightarrow$ Hidden state
- Statistical $\leftrightarrow$ Adversarial

Deterministic shortest-path problems

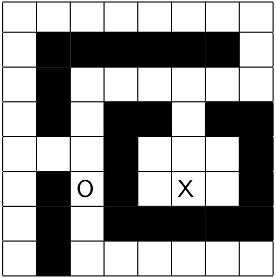


Properties

- $g(t) = 1, \; T \to \infty$.
- $r_t = -1$ unless $s_t = X$, in which case $r_t = 0$.
- $\mu(s_{t+1} = X | s_t = X) = 1$.
- $\mathcal{A} = \{\mathrm{North}, \mathrm{South}, \mathrm{East}, \mathrm{West}\}$
- Transitions are deterministic and walls block.

What is the shortest path to the destination from any point?

Stochastic shortest path problem, with a pit



Properties

- $g(t) = 1, \; T \to \infty$.
- $r_t = -1$, but $r_t = 0$ at X and -100 at O and episode ends.
- $\mu(s_{t+1} = X | s_t = X) = 1$.
- $\mathcal{A} = \{\text{North}, \text{South}, \text{East}, \text{West}\}$
- Moves to a random direction with probability θ. Walls block.

For what value of θ is it better to take the dangerous shortcut? (However, if we want to take into account risk explicitly we must modify the agent's utility function)

Continuing stochastic MDPs

Inventory management

- There are K storage locations.
- Each place can store n_i items.
- At each time-step there is a probability ϕ_i that a client try to buy an item from location i, $\sum_i \phi_i \leq 1$. If there is an item available, you gain reward 1.
- Action 1: ordering u units of stock, for paying $c(u)$.
- Action 2: move u units of stock from one location i to another, j, for a cost $\psi_{ij}(u)$.

An easy special case

- $K = 1$.
- There is one type of item only.
- Orders are placed and received every n timesteps.

Inventory management

An easy special case

- $K = 1$.
- Deliveries happen once every m timesteps.
- Each time-step a client arrives with probability ϕ.

Properties

- The state set .
- The action set .
- The transition probabilities

Inventory management

An easy special case

- $K = 1$.
- Deliveries happen once every m timesteps.
- Each time-step a client arrives with probability ϕ.

Properties

- The state set is the number of items we have: $\mathcal{S} = \{0, 1, \ldots, n\}$.
- The action set .
- The transition probabilities

Inventory management

An easy special case

- $K = 1$.
- Deliveries happen once every m timesteps.
- Each time-step a client arrives with probability ϕ.

Properties

- The state set is the number of items we have: $\mathcal{S} = \{0, 1, \ldots, n\}$.
- The action set $\mathcal{A} = \{0, 1, \ldots, n\}$ since we can order from nothing up to n items.
- The transition probabilities

Inventory management

An easy special case

- $K = 1$.
- Deliveries happen once every m timesteps.
- Each time-step a client arrives with probability ϕ.

Properties

- The state set is the number of items we have: $\mathcal{S} = \{0, 1, \ldots, n\}$.
- The action set $\mathcal{A} = \{0, 1, \ldots, n\}$ since we can order from nothing up to n items.
- The transition probabilities $P(s'|s, a) = \binom{m}{d}\phi^d(1-\phi)^{m-d}$, where $d = s + a - s'$, for $s + a \leq n$.

Episodic, finite, infinite?

Shortest path problems

- Episodic tasks with infinite horizon, -1 reward everywhere, but 0 in absorbing state.
- Continuing tasks with 0 reward everywhere, but > 0 in goal state, $\gamma \in (0, 1)$, state reset after goal.
- Equivalent if optimal policy is the same.

Introduction

Why dynamic programming?

- Programming means finding a solution.
- i.e. linear programming.
- Dynamic because we find solution to dynamical problems.
- Direct relation to control theory.

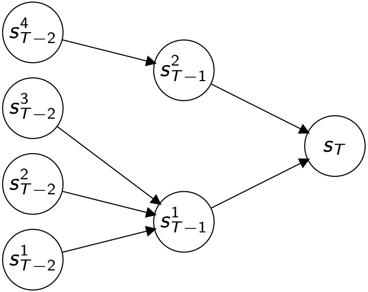
The shortest-path problem revisited

14	13	12	11	10	9	8	7
15		13					6
16	15	14		4	3	4	5
17					2		
18	19	20		2	1	2	
19		21		1	0	1	
20		22					
21		23	24	25	26	27	28

Properties

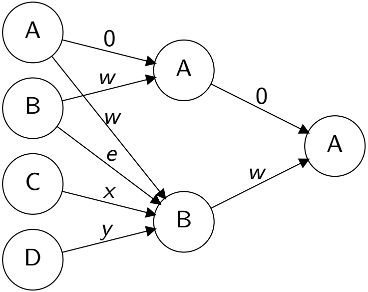
- $\gamma = 1, \ T \to \infty$.
- $r_t = -1$ unless $s_t = X$, in which case $r_t = 0$.
- The length of the shortest path from s equals the negative value of the optimal policy.
- Also called *cost-to-go*.
- Remember Dijkstra's algorithm?

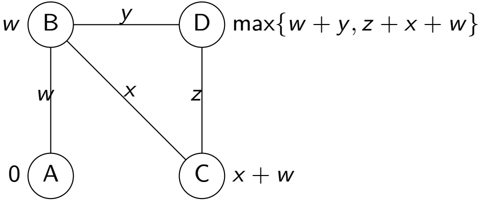
Backwards induction I



- If we know the value of the last state, we can calculate the values of its predecessors.
- The value of s^i_{T-1} is the reward obtained by moving from s^i_{T-1} to s_T, plus the value of s_T.

Backwards induction II





- All $w, x, y, z < 0$, and reward $e < 0$ of staying at the same state, apart from A.
- All w, x, y, z

Backwards induction III

Backwards induction in deterministic environments

Input μ, $\mathcal{S}_T$.
Initialise $V_T(s)$, for all $s \in \mathcal{S}_T$.
for $n = T-1, T-2, \ldots, t$ **do**
  for $s \in \mathcal{S}_n$ **do**
    $a_n^*(s) = \arg\max_a \mathbf{E}(r | s'_{s,a}, s, \mu) + V_{n+1}^*(s'_{s,a})$
    $V_n^*(s) = \mathbf{E}(r | s'_{s,a_n^*(s)}, s, \mu) + V_{n+1}^*(s'_{s,a_n^*(s)})$
  end for
end for

Notes

- $s'_{s,a}$ is the state that occurs if we take a in s.
- Because we always know the optimal choice at the last step, we can find the optimal policy directly!

Backwards induction III

Backwards induction in deterministic environments

Input μ, $\mathcal{S}_T$.
Initialise $V_T(s)$, for all $s \in \mathcal{S}_T$.
for $n = T-1, T-2, \ldots, t$ **do**
  for $s \in \mathcal{S}_n$ **do**
    $a_n^*(s) = \arg\max_a \sum_{s' \in \mathcal{S}_{n+1}} \mu(s'|s, a)\, \mathbf{E}(r|s', s, \mu) + V_{n+1}^*(s')$
    $V_n(s)^* = \sum_{s' \in \mathcal{S}_{n+1}} \mu(s'|s, a_n^*(s))\, \mathbf{E}(r|s', s, \mu) + V_{n+1}^*(s')$
  end for
end for

Notes

- $\mu(s'|s, a)$ is an *indicator function*
- Because we always know the optimal choice at the last step, we can find the optimal policy directly!

Backwards induction III

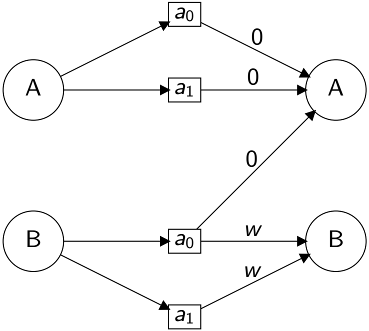
Backwards induction in deterministic environments

Input μ, $\mathcal{S}_T$.
Initialise $V_T(s)$, for all $s \in \mathcal{S}_T$.
for $n = T - 1, T - 2, \ldots, t$ **do**
  for $s \in \mathcal{S}_n$ **do**
    $a_n^*(s) = \arg\max_a \sum_{s' \in \mathcal{S}_{n+1}} \mu(s'|s, a) \, \mathbf{E}(r|s', s, \mu) + V_{n+1}^*(s')$
    $V_n(s)^* = \sum_{s' \in \mathcal{S}_{n+1}} \mu(s'|s, a_n^*(s)) \, \mathbf{E}(r|s', s, \mu) + V_{n+1}^*(s')$
  end for
end for

Notes

- $\mu(s'|s, a)$ is an *indicator function*
- Nothing apparently stops $\mu(s'|s, a)$ from being a distribution
- So, what happens in stochastic environments?

Backwards induction IV: Stochastic problems



Almost as before, but state depends stochastically on actions, i.e. $\mu(s_{t+1}{=}A|s_t{=}B, a_t{=}a)$

The backup operators

$$V_n^{\pi}(s) = \sum_{s'} [\mu(s'|s, \pi)\, \mathbf{E}(r|s', s) + V_{n+1}^{\pi}(s')] \tag{14}$$

$$V_n^{*}(s) = \max_a \sum_{s'} \mu(s'|s, a)[\mathbf{E}(r|s', s) + V_{n+1}^{*}(s')] \tag{15}$$

Backwards induction V

Policy evaluation with Backwards induction

Input π, μ, $\mathcal{S}_T$.
Initialise $V_T(s)$, for all $s \in \mathcal{S}_T$.
for $n = T-1, T-2, \ldots, t$ **do**
  for $s \in \mathcal{S}_n$ **do**
    $V_n^\pi(s) = \sum_{s' \in \mathcal{S}_{n+1}} \mu(s'|s,\pi)[\mathbf{E}(r|s', s, \mu) + V_{n+1}^\pi(s')]$
  end for
end for

Notes

- $\mu(s'|s,\pi) = \sum_a \mu(s'|s,a)\pi(a|s)$.
- Finite horizon problems only, or approximations to finite horizon (i.e. lookahead in game trees).
- Hey, it works for stochastic problems too! (By marginalizing over states)
- Because we always know the optimal choice at the last step, we can find the optimal policy directly!
- Can be used with estimates of the value function.

Backwards induction V

Finding the optimal policy with Backwards induction

Input μ, $\mathcal{S}_T$.
Initialise $V_T(s)$, for all $s \in \mathcal{S}_T$.
for $n = T-1, T-2, \ldots, t$ **do**
  for $s \in \mathcal{S}_n$ **do**
    $a_n^*(s) = \arg\max_a \mu(s'|s,a)[\mathbf{E}(r|s',s,\mu) + V_{n+1}^*(s')]$
    $V_n(s)^* = \sum_{s' \in \mathcal{S}_{n+1}} \mu(s'|s,a_n^*)[\mathbf{E}(r|s',s,\mu) + V_{n+1}^*(s')]$
  end for
end for

Notes

- Finite horizon problems only, or approximations to finite horizon (i.e. lookahead in game trees).
- Hey, it works for stochastic problems too! (By marginalizing over states)
- Because we always know the optimal choice at the last step, we can find the optimal policy directly!
- Can be used with estimates of the value function.

Infinite horizon

What happens when the horizon is infinite in stochastic shortest path problems?

- Episodic tasks still terminate with probability one for proper policies.
- Assumption: there exists at least one proper policy.
- Assumption: Every improper policy has negatively infinite value for at least one state.

Policy improvement

Why evaluate a policy?

We can always generate a better policy given the value function of any policy!

Theorem (Policy improvement)

Let some policy $\pi \in \mathcal{P}$*. If* $\pi'(a|s) = 1$ *for* $a = \arg\max_{a'} Q^{\pi}(s, a)$ *and* 0 *otherwise, then*

$$V^{\pi'}(s) \geq V^{\pi}(s), \quad \forall s \in \mathcal{S}$$

Policy improvement theorem

Theorem (Policy improvement)

Let some policy $\pi \in \mathcal{P}$. *If* $\pi'(a|s) = 1$ *for* $a = \arg\max_{a'} Q^{\pi}(s, a)$ *and* 0 *otherwise, then*

$$V^{\pi'}(s) \geq V^{\pi}(s), \quad \forall s \in \mathcal{S}$$

Proof.

Let π_k be the policy which execute π' for k steps and then reverts to π. Then $\pi = \pi_0$, $\pi' = \lim_{k\to\infty} \pi_k$, and we have

$$V^{\pi}(s_t) = \sum_{a_t} \pi(a_t|s_t) Q^{\pi}(s, a)$$

$$\leq \max_{a_t} Q^{\pi}(s, a) = \max_{a_t} \left[\sum_{s_{t+1}} \mu(s_{t+1}|s_t, a_t) V^{\pi}(s_{t+1}) \right] = V^{\pi_1}(s_t).$$

Similarly, we show that $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$ for all s. Then $V^{\pi} \leq V^{\pi_1}(s) \leq V^{\pi_k}(s) \leq V^{\pi_{k+1}}(s) \dots$ and so $V^{\pi'}(s) = \lim_{k\to\infty} V^{\pi_k}(s) \geq V^{\pi}(s)$. $\square$

Iterative policy evaluation

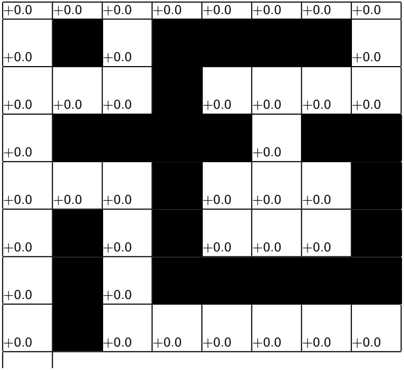
Policy Evaluation

Input π, μ and $\hat{V}_0$.
$n = 0$.
repeat
  $n = n + 1$
  for $s \in S$ **do**
    $\hat{V}_n(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \mu(s'|s,a)[\mathbf{E}(r|s',\mu) + \gamma \hat{V}_{n-1}(s')]$
  end for
until $\|\hat{V}_n - \hat{V}_{n-1}\|_\infty < \theta$

Notes

- Arbitrary initialization.
- $V^\pi, \hat{V}_n \in \mathbb{R}^{|\mathcal{S}|}$,
- $\lim_{n\to\infty} \hat{V}_n = V^\pi$, if the limit exists.
- Can be done in-place as well.

Policy evaluation example I



0 iterations

Random policy evaluation.

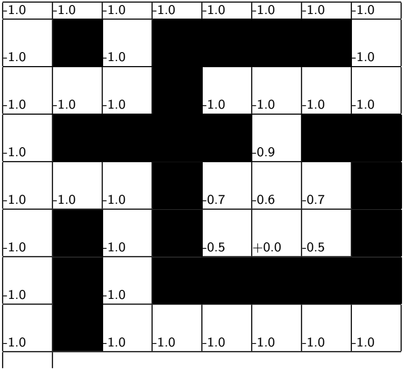
Policy evaluation example I

-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1
-0.1		-0.1					-0.1
-0.1	-0.1	-0.1		-0.1	-0.1	-0.1	-0.1
-0.1					-0.1		
-0.1	-0.1	-0.1		-0.1	-0.1	-0.1	
-0.1		-0.1		-0.1	+0.0	-0.1	
-0.1		-0.1					
-0.1		-0.1	-0.1	-0.1	-0.1	-0.1	-0.1

1 iteration

Random policy evaluation.

Policy evaluation example I



Random policy evaluation.

10 iterations

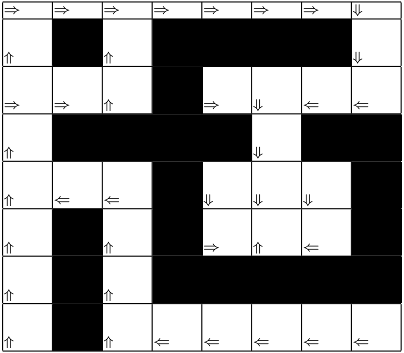
Policy evaluation example I

-9.8	-9.8	-9.7	-9.6	-9.4	-9.2	-8.9	-8.5
-9.8		-9.8					-8.0
-9.9	-9.8	-9.8		-5.7	-5.4	-6.5	-7.4
-9.9					-3.8		
-9.9	-9.9	-9.9		-1.6	-1.8	-1.6	
-9.9		-9.9		-1.0	+0.0	-1.0	
-9.9		-9.9					
-9.9		-9.9	-9.9	-9.9	-9.9	-9.9	-9.9

Random policy evaluation.

99 iterations

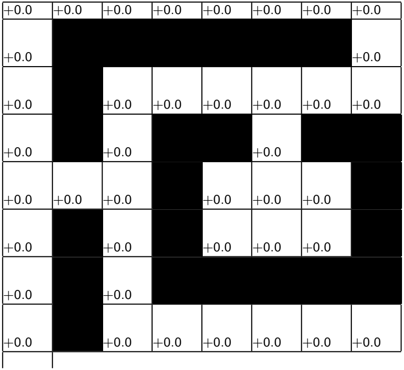
Policy evaluation example I

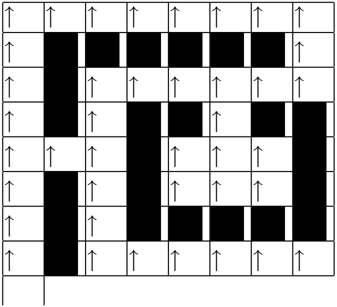


Random policy evaluation.

Greedy policy with respect to value function of random policy

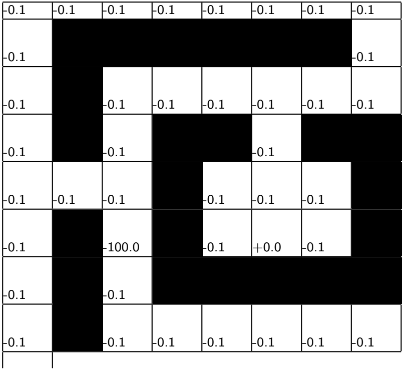
Policy evaluation example II

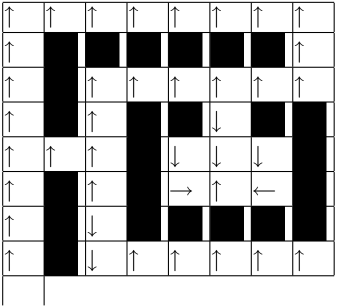




Random policy evaluation.

Policy evaluation example II





Random policy evaluation.

Policy evaluation example II

-1.1	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0	-1.0
-1.3							-1.0
-2.3		-14.1	-6.1	-2.6	-1.3	-1.0	-1.0
-5.1		-28.7			-0.9		
-11.1	-27.7	-50.4		-0.7	-0.6	-0.7	
-5.1		-100.0		-0.5	+0.0	-0.5	
-2.3		-65.3					
-1.4		-36.7	-17.6	-7.3	-2.9	-1.4	-1.1

→	→	→	↑	→	→	→	↓
↑							→
↑		→	→	→	↓	→	↑
↑		↑			↓		
↑	←	←		↓	↓	↓	
↓		↑		→	↑	←	
↓		↓					
↓		→	→	→	→	→	↑

Random policy evaluation.

Policy evaluation example II

-31.3	-27.2	-24.0	-21.5	-19.8	-18.8	-18.5	-18.7
-36.2							-19.4
-41.9		-55.8	-44.9	-34.2	-23.6	-22.0	-20.5
-48.5		-66.7			-14.8		
-55.9	-66.7	-77.8		-4.3	-5.9	-4.3	
-53.1		-100.0		-2.3	+0.0	-2.3	
-51.2		-93.0					
-50.2		-86.0	-79.5	-73.8	-69.2	-66.0	-64.3

Random policy evaluation.

$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\uparrow$	$\leftarrow$
$\uparrow$							$\uparrow$
$\uparrow$		$\rightarrow$	$\rightarrow$	$\rightarrow$	$\downarrow$	$\rightarrow$	$\uparrow$
$\uparrow$		$\uparrow$			$\downarrow$		
$\uparrow$	$\leftarrow$	$\leftarrow$		$\downarrow$	$\downarrow$	$\downarrow$	
$\downarrow$		$\uparrow$		$\rightarrow$	$\uparrow$	$\leftarrow$	
$\downarrow$		$\downarrow$					
$\downarrow$		$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\uparrow$

Value iteration

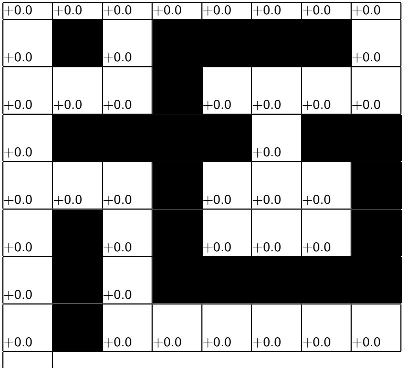
Value Iteration

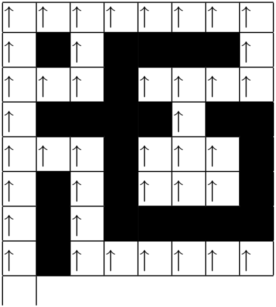
Input μ.
$\hat{V}_0(s) = 0$ for all $s \in \mathcal{S}$.
$n = 0$.
repeat
  $n = n + 1$
  for $s \in S$ **do**
    $\hat{V}_n(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mu(s'|s, a)[\mathbf{E}(r|s', \mu) + \gamma \hat{V}_{n-1}(s')]$
  end for
until $\|\hat{V}_n - \hat{V}_{n-1}\|_\infty < \theta$

Notes

- No reason to assume a fixed policy, convergence holds.
- $\lim_{n \to \infty} \hat{V}_n = V^*$.
- Equivalent to backwards induction as horizon $\to \infty$.
- This is because $\lim T \to \infty V_t^\pi(s) = V^\pi(s)$ for all t.

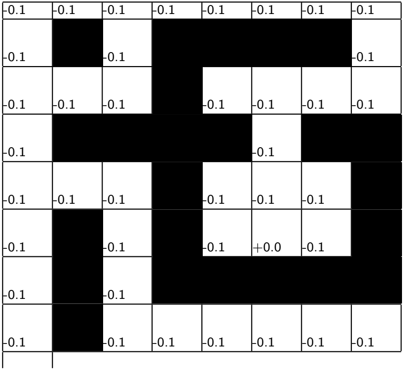
Value iteration example



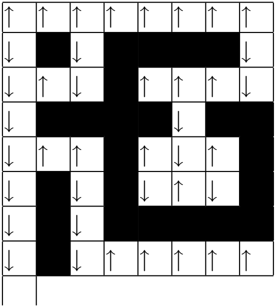


iter: 0

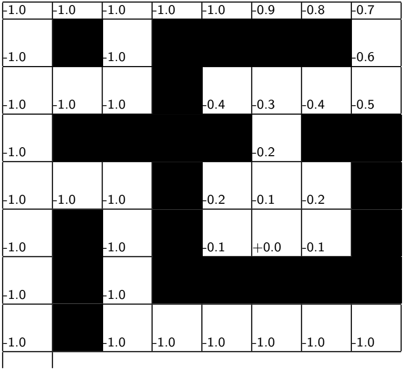
Value iteration example



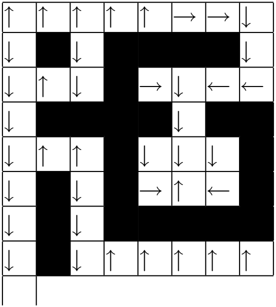
iter: 1



Value iteration example



iter: 10



Value iteration example

-1.4	-1.3	-1.2	-1.1	-1.0	-0.9	-0.8	-0.7
-1.5		-1.3					-0.6
-1.6	-1.5	-1.4		-0.4	-0.3	-0.4	-0.5
-1.7					-0.2		
-1.8	-1.9	-2.0		-0.2	-0.1	-0.2	
-1.9		-2.1		-0.1	+0.0	-0.1	
-2.0		-2.2					
-2.1		-2.3	-2.4	-2.5	-2.6	-2.7	-2.8

iter: 100

→	→	→	→	→	→	→	↓
↑		↑					↓
↑	→	↑		→	↓	←	←
↑					↓		
↑	←	←		↓	↓	↓	
↑		↑		→	↑	←	
↑		↑					
↑		↑	←	←	←	←	←

Policy iteration I

Policy Iteration

Input π, μ.
repeat
  Evaluate V^{π}.
  $\pi' : \pi'(s) = \arg\max_a Q^{\pi}(s, a)$
until $\arg\max_a Q^{\pi'}(s, a) = V^{\pi}(s)$ for all s

Theorem (Policy iteration)

The policy iteration algorithm generates an improving sequence of proper policies, i.e.

$$V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s), \quad \forall k > 0, s \in \mathcal{S}$$

and terminates with an optimal policy, i.e. $\lim_{k\to\infty} V^{\pi_k} = V^*$.

Remark (Policy iteration termination)

If π_k *is not optimal, then* $\exists s \in \mathcal{S}$:

$$V^{\pi_{k+1}}(s) > V^{\pi_k}(s).$$

Conversely, if no such s *exists,* π_k *is optimal and we terminate.*

Policy iteration II

The evaluation step

- It can be done exactly by solving the linear equations. (Proper policy iteration)
- We can use a limited number n of policy evaluation iterations (Modified policy iteration algorithm).
- These can be initalised from the last evaluation.
- If we use just $n = 1$, then the method is identical to value iteration.
- If we use $n \to \infty$, then we have proper policy iteration.

Other methods

- Asynchronous policy iteration.
- Multistage lookahead policy iteration.
- See [1], section 2.2 for more details.
- See [3], Chapters 4,5,6 for detailed theory.

Lessons learnt

Planning with a known model

- Find the optimal policy given model and objective.
- Bellman recursion is the basis of dynamic programming.
- Easy to solve for finite-horizon problems or episodic tasks.
- Stochasticity does not make the problem significantly harder.
- Infinite-horizon continuing problems harder, but tractable.

Things to think about

- Would iterative methods be better than backwards induction?
- How does it depend on the problem?
- Does the discount factor have any effect?
- How can backwards induction be applied to iterative problems and vice-versa?

Learning from reinforcement...

Bandit problems

- $\gamma \in [0, 1]$, $T > 0$.
- $|\mathcal{S}| = 1$.
- Rewards are random with expectation $\mathbf{E}[r_t | a_t, \mu]$
- If μ known, trivial: $a^* = \arg\max_a \mathbf{E}[r_t | a_t = a, \mu]$, for all t, γ.
- If μ is unknown, can be intractable.
- Simplest case of *learning from reinforcement*.

Further reading

- Dimitri P. Bertsekas and John N. Tsitsiklis.
  Neuro-Dynamic Programming.
  Athena Scientific, 1996.
- Morris H. DeGroot.
  Optimal Statistical Decisions.
  John Wiley & Sons, 1970.
  Republished in 2004.
- Marting L. Puterman.
  Markov Decision Processes : Discrete Stochastic Dynamic Programming.
  John Wiley & Sons, New Jersey, US, 1994,2005.
- Richard S. Sutton and Andrew G. Barto.
  Reinforcement Learning: An Introduction.
  MIT Press, 1998.