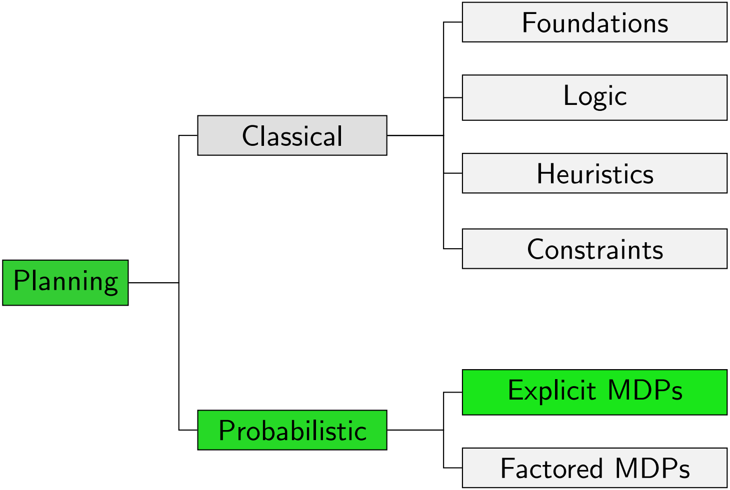
Planning and Optimization

F3. Policy Iteration

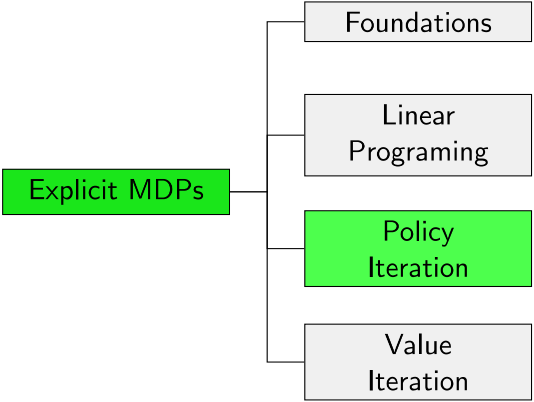
Malte Helmert and Gabriele Röger

Universität Basel

Content of this Course



Content of this Course: Explicit MDPs



Introduction

Limitations of LPs in Practice

With the LP we can compute an optimal policy in polynomial time.

Possible issues in practice:

- LPs often too expensive even for small MDPs
- LP solver usage prohibited
- More expressive model required (e.g. continuous state space)

Limitations of LPs in Practice

With the LP we can compute an optimal policy in polynomial time.

Possible issues in practice:

- LPs often too expensive even for small MDPs
- LP solver usage prohibited
- More expressive model required (e.g. continuous state space)

Policy Iteration (PI) is a suitable alternative.
It has 2 components:

- Policy Evaluation: Compute V_π for a given π
- Policy Improvement: Determine better policy from V_π

Policy Evaluation

Reminder: Value Functions for SSPs

Definition (Value Functions for SSPs)

Let $\mathcal{T} = \langle S, A, c, T, s_0, S_\star \rangle$ be an SSP and π be a policy for $\mathcal{T}$.

The state-value $V_\pi(s)$ of s under π is defined as

$$V_\pi(s) := \begin{cases} 0 & \text{if } s \in S_\star \\ Q_\pi(s, \pi(s)) & \text{otherwise,} \end{cases}$$

where the action-value $Q_\pi(s, a)$ of s and a under π is defined as

$$Q_\pi(s, a) := c(a) + \sum_{s' \in \text{succ}(s,a)} T(s, a, s') \cdot V_\pi(s').$$

The state-value $V_\pi(s)$ describes the expected cost
of applying π in SSP $\mathcal{T}$, starting from s.

Policy Evaluation: Implementations

Computing V_π for a given policy π is called policy evaluation.

There are several algorithms for policy evaluation:

1. Linear Program

Reminder: LP for Expected Cost in SSP

Variables

Non-negative variable ExpCost_s for each state s

Objective

Maximize ExpCost_{s_0}

Subject to

$$\text{ExpCost}_{s_\star} = 0 \quad \text{for all goal states } s_\star$$

$$\text{ExpCost}_s \leq \Big(\sum_{s' \in S} T(s,a,s') \cdot \text{ExpCost}_{s'}\Big) + c(a)$$

$$\text{for all } s \in S \text{ and } a \in A(s)$$

LP for Policy Evaluation in SSP

Variables

Non-negative variable ExpCost_s for each state s

Objective

Maximize ExpCost_{s_0}

Subject to

$$\text{ExpCost}_{s_\star} = 0 \quad \text{for all goal states } s_\star$$

$$\text{ExpCost}_s \leq \Big(\sum_{s' \in S} T(s, \pi(s), s') \cdot \text{ExpCost}_{s'}\Big) + c(\pi(s))$$

$$\text{for all } s \in S \ \sout{\text{and } a \in A(s)}$$

Policy Evaluation via LP

- is polynomial in $|S|$
- difference between polynomial in $|S|$ and polynomial in $|S| \cdot |A|$ is sometimes relevant in practice
- but often this is not the case
- other practical limitations also apply here

$\rightsquigarrow$ Require policy evaluation without LP

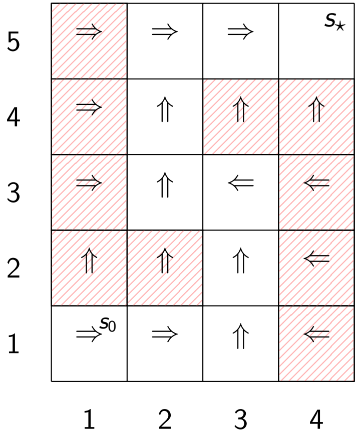
Policy Evaluation: Implementations

Computing V_π for a given policy π is called policy evaluation.

There are several algorithms for policy evaluation:

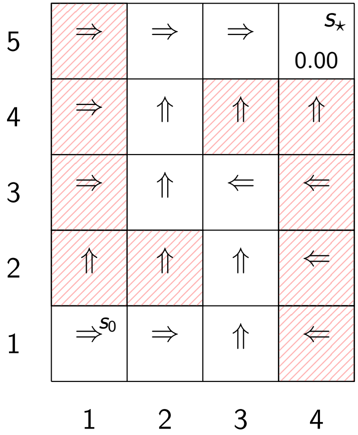
1. Linear Program
2. Backward Induction

Example: Backward Induction in Deterministic SSP



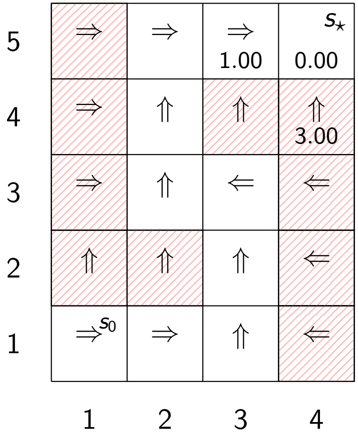
- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



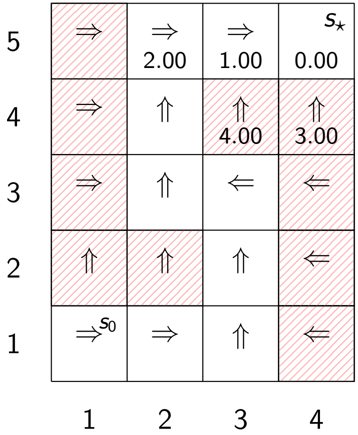
- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



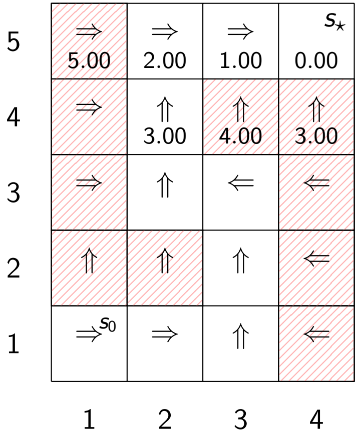
- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



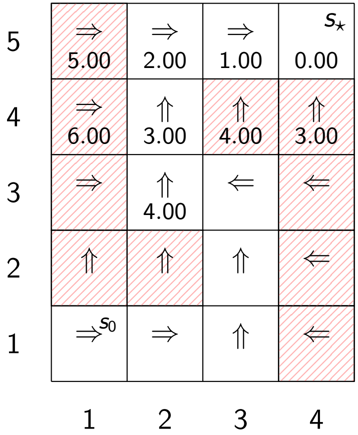
- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



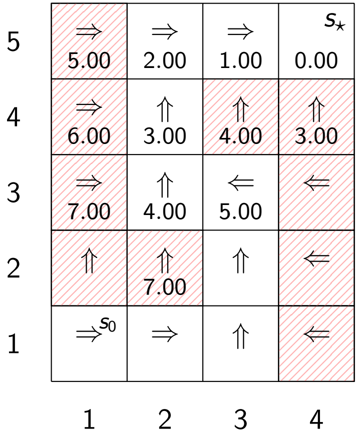
- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



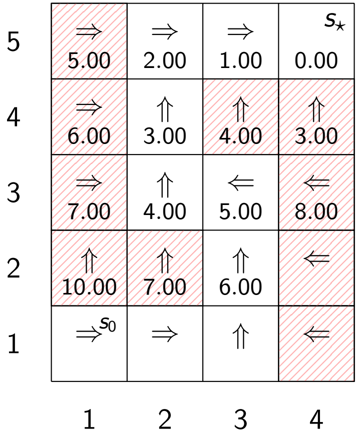
- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



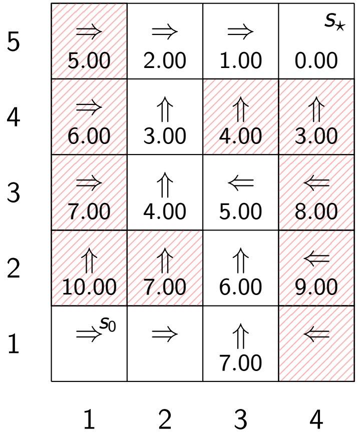
- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



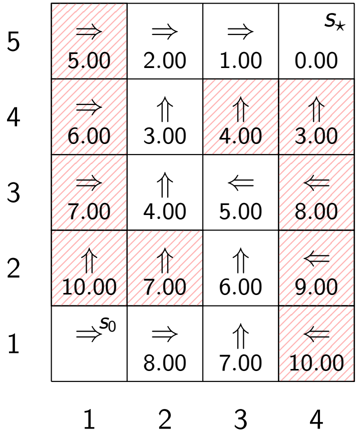
- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



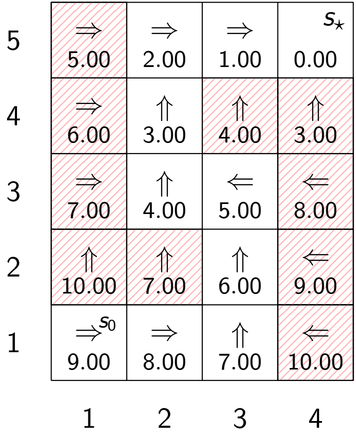
- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



- cost of 3 to move from striped cells (cost is 1 otherwise)

Example: Backward Induction in Deterministic SSP



- cost of 3 to move from striped cells (cost is 1 otherwise)

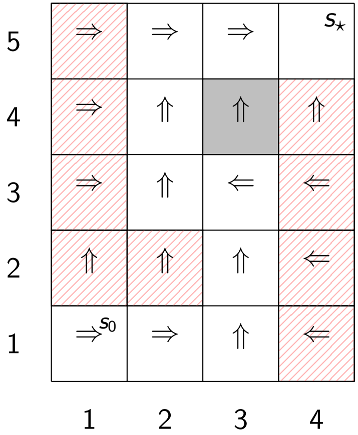
Policy Evaluation via Backward Induction

- is linear in $|S|$
- but restricted to special cases

$\rightsquigarrow$ When is policy evaluation via backward induction possible?

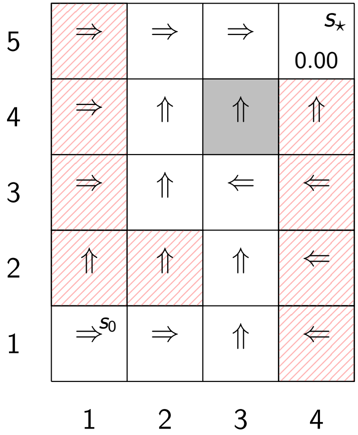
In deterministic planning problems?

Example: Backward Induction in Probabilistic SSP



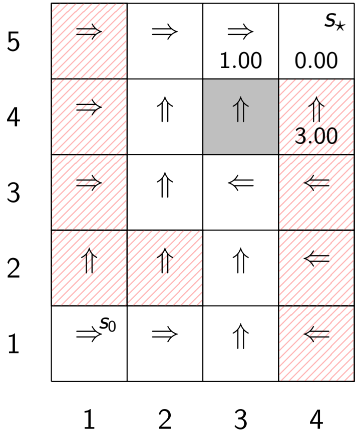
- cost of 3 to move from striped cells (cost is 1 otherwise)
- probability of 0.4 to "⇒" in gray cell

Example: Backward Induction in Probabilistic SSP



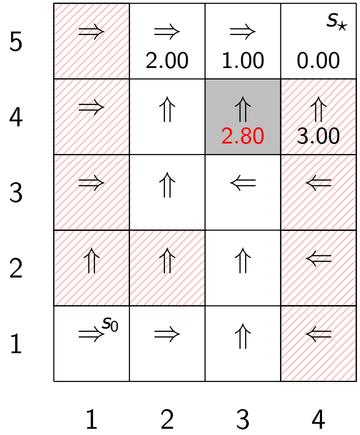
- cost of 3 to move from striped cells (cost is 1 otherwise)
- probability of 0.4 to "⇒" in gray cell

Example: Backward Induction in Probabilistic SSP



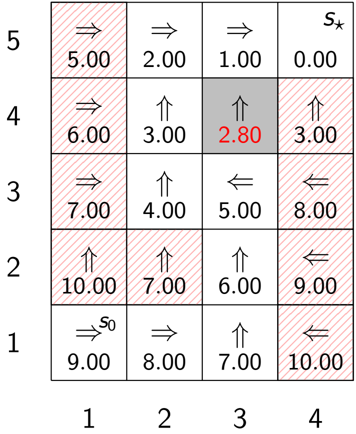
- cost of 3 to move from striped cells (cost is 1 otherwise)
- probability of 0.4 to "⇒" in gray cell

Example: Backward Induction in Probabilistic SSP



- cost of 3 to move from striped cells (cost is 1 otherwise)
- probability of 0.4 to "⇒" in gray cell

Example: Backward Induction in Probabilistic SSP



- cost of 3 to move from striped cells (cost is 1 otherwise)
- probability of 0.4 to "$\Rightarrow$" in gray cell

Policy Evaluation via Backward Induction

$\rightsquigarrow$ When is policy evaluation via backward induction possible?

In deterministic planning problems?

No, policy must be acyclic.

Policy Evaluation: Implementations

Computing V_π for a given policy π is called policy evaluation.

There are several algorithms for policy evaluation:

1. Linear Program
2. Backward Induction for acyclic policies

Backward Induction: Algorithm

Backward Induction for SSP $\langle S, A, c, T, s_0, S_\star \rangle$ and complete policy π

initialize $V_\pi(s) :=$ none for all $s \in S$
$V_\pi(s) := 0$ for all $s \in S_\star$
while there is a $s \in S$ with $V_\pi(s) =$ none:
  pick $s \in S$ with $V_\pi(s) =$ none and
    $V_\pi(s') \neq$ none for all $s' \in \mathrm{succ}(s, \pi(s))$
  set $V_\pi(s) := c(\pi(s)) + \sum_{s' \in S} T(s, \pi(s), s') \cdot V_\pi(s')$
return V_π

Policy Evaluation: Implementations

Computing V_π for a given policy π is called policy evaluation.

There are several algorithms for policy evaluation:

1. Linear Program
2. Backward Induction for acyclic policies
3. Iterative Policy Evaluation

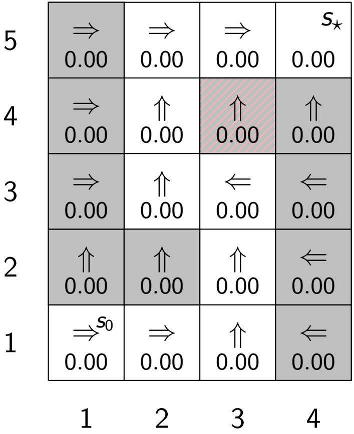
Iterative Policy Evaluation: Idea

- impossible to compute state-values in one sweep over the state space in presence of cycles
- start with arbitrary state-value function $\hat{V}_\pi^0$
- treat state-value function as update rule

$$\hat{V}_\pi^i(s) = c(\pi(s)) + \sum_{s' \in S} T(s, \pi(s), s') \cdot \hat{V}_\pi^{i-1}(s')$$

- apply update rule iteratively
- until state-values have converged

Iterative Policy Evaluation for SSPs: Example



$\hat{V}_\pi^0$

- cost of 3 to move from striped cells (cost is 1 otherwise)
- moving from gray cells unsuccessful with probability 0.6

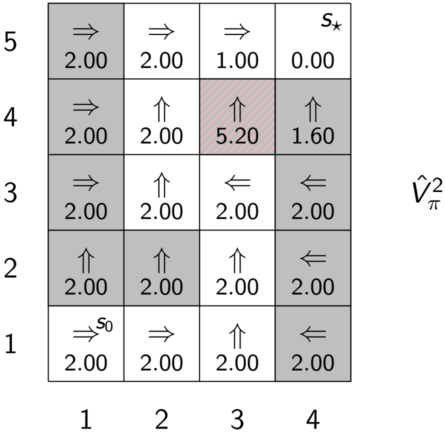
Iterative Policy Evaluation for SSPs: Example

	1	2	3	4
5	⇒ 1.00	⇒ 1.00	⇒ 1.00	$s_\star$ 0.00
4	⇒ 1.00	⇑ 1.00	⇑ 3.00	⇑ 1.00
3	⇒ 1.00	⇑ 1.00	⇐ 1.00	⇐ 1.00
2	⇑ 1.00	⇑ 1.00	⇑ 1.00	⇐ 1.00
1	⇒ s_0 1.00	⇒ 1.00	⇑ 1.00	⇐ 1.00

$\hat{V}_\pi^1$

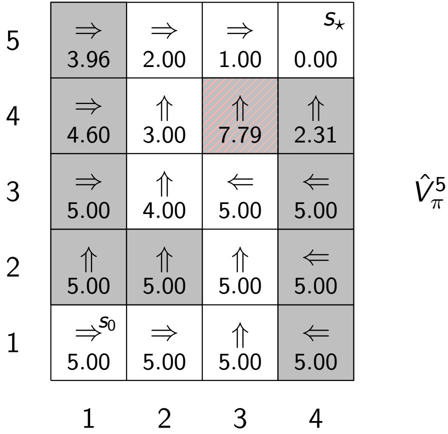
- cost of 3 to move from striped cells (cost is 1 otherwise)
- moving from gray cells unsuccessful with probability 0.6

Iterative Policy Evaluation for SSPs: Example



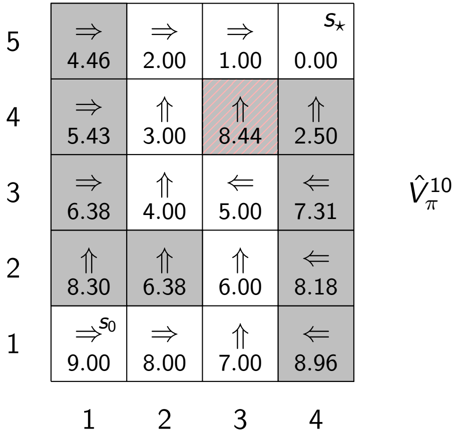
- cost of 3 to move from striped cells (cost is 1 otherwise)
- moving from gray cells unsuccessful with probability 0.6

Iterative Policy Evaluation for SSPs: Example



- cost of 3 to move from striped cells (cost is 1 otherwise)
- moving from gray cells unsuccessful with probability 0.6

Iterative Policy Evaluation for SSPs: Example



- cost of 3 to move from striped cells (cost is 1 otherwise)
- moving from gray cells unsuccessful with probability 0.6

Iterative Policy Evaluation for SSPs: Example

	1	2	3	4
5	$\Rightarrow$ 4.50	$\Rightarrow$ 2.00	$\Rightarrow$ 1.00	$s_\star$ 0.00
4	$\Rightarrow$ 5.50	$\Uparrow$ 3.00	$\Uparrow$ 8.50	$\Uparrow$ 2.50
3	$\Rightarrow$ 6.50	$\Uparrow$ 4.00	$\Leftarrow$ 5.00	$\Leftarrow$ 7.50
2	$\Uparrow$ 9.00	$\Uparrow$ 6.50	$\Uparrow$ 6.00	$\Leftarrow$ 8.50
1	$\Rightarrow$ s_0 9.00	$\Rightarrow$ 8.00	$\Uparrow$ 7.00	$\Leftarrow$ 9.50

$\hat{V}_\pi^{29}$

- cost of 3 to move from striped cells (cost is 1 otherwise)
- moving from gray cells unsuccessful with probability 0.6

Iterative Policy Evaluation: Algorithm

Iterative Policy Evaluation for SSP $\langle S, A, c, T, s_0, S_\star \rangle$, complete policy π and $\epsilon > 0$

initialize $\hat{V}^0$ to 0 for goal states, otherwise arbitarily
for $i = 1, 2, \ldots$:
 for all states $s \in S \setminus S_\star$:
  $\hat{V}^i_\pi(s) := c(\pi(s)) + \sum_{s' \in S} T(s, \pi(s), s') \cdot \hat{V}^{i-1}_\pi(s')$
 if $\max_{s \in S} |\hat{V}^i_\pi(s) - \hat{V}^{i-1}_\pi(s)| < \epsilon$:
  return $\hat{V}^i_\pi$

Iterative Policy Evaluation: Properties

Theorem (Convergence of Iterative Policy Evaluation)

Let $\mathcal{T} = \langle S, A, c, T, s_0, S_\star \rangle$ be an SSP, π be a proper policy for $\mathcal{T}$ and $\hat{V}^0_\pi(s) \in \mathbb{R}$ arbitrarily for all $s \setminus S_\star$.

Iterative policy evaluation converges to the true state-values, i.e.,

$$\lim_{i \to \infty} \hat{V}^i_\pi(s) = V_\pi(s) \text{ for all } s \in S.$$

Proof omitted.

In practice, iterative policy evaluation converges to true state-values if ϵ is small enough.

Policy Evaluation: MDPs

What about policy evaluation for MDPs?

- MDPs (with finite state set) are always cyclic
  $\Rightarrow$ backward induction not applicable
- but goal state not required for iterative policy evaluation
- albeit traces are infinite, iterative policy evaluation converges
- convergence theorem also holds for MDPs

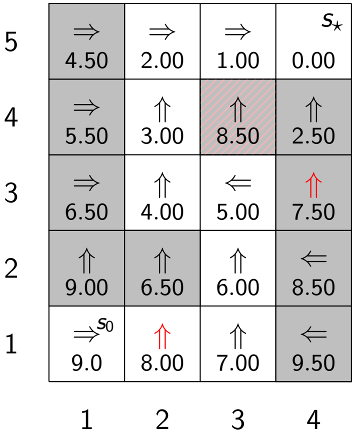
Policy Improvement

Example: Greedy Action

	1	2	3	4
5	⇒ 4.50	⇒ 2.00	⇒ 1.00	$s_\star$ 0.00
4	⇒ 5.50	⇑ 3.00	⇑ 8.50	⇑ 2.50
3	⇒ 6.50	⇑ 4.00	⇐ 5.00	⇐ 7.50
2	⇑ 9.00	⇑ 6.50	⇑ 6.00	⇐ 8.50
1	⇒ s_0 9.0	⇒ 8.00	⇑ 7.00	⇐ 9.50

- Can we learn more from this than the state-values of a policy?

Example: Greedy Action



- Can we learn more from this than the state-values of a policy?
- Yes! By evaluating all actions in each state, we can derive a better policy

Greedy Actions and Policies for SSPs

Definition (Greedy Action)

Let s be a state of an SSP $\mathcal{T} = \langle S, A, c, T, s_0, S_\star \rangle$ and V be a state-value function for $\mathcal{T}$.
The set of greedy actions in s with respect to V is

$$A_V(s) := \arg \min_{a \in A(s)} \left(c(a) + \sum_{s' \in S} T(s, a, s') \cdot V(s') \right).$$

A policy π_V with $\pi_V(s) \in A_V(s)$ is a greedy policy.

Determining a greedy policy of a given state-value function is called policy improvement.

Greedy Actions and Policies for MDPs

Definition (Greedy Action)

Let s be a state of a (discounted-reward) MDP $\mathcal{T} = \langle S, A, R, T, s_0, \gamma \rangle$ and V be a state-value function for $\mathcal{T}$. The set of greedy actions in s with respect to V is

$$A_V(s) := \arg\max_{a \in A(s)} \left(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \cdot V(s') \right).$$

A policy π_V with $\pi_V(s) \in A_V(s)$ is a greedy policy.

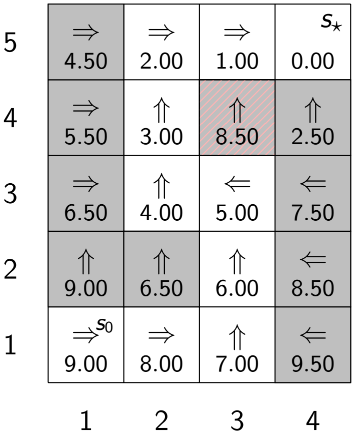
Determining a greedy policy of a given state-value function is called policy improvement.

Policy Iteration

Policy Iteration

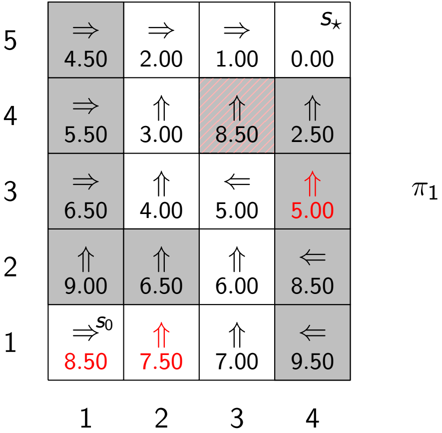
- Policy Iteration (PI) was first proposed by Howard in 1960
- based on the observation that the greedy actions describe a better policy
- starts with arbitrary policy π_0
- alternates policy evaluation and policy improvement
- as long as policy changes

Example: Policy Iteration

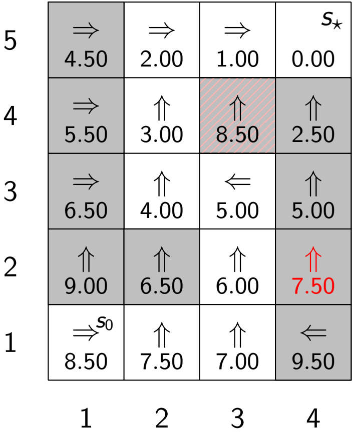


π_0

Example: Policy Iteration



Example: Policy Iteration



$\pi_2 = \pi_3$

Policy Iteration: Algorithm

Policy Iteration for SSP or MDP $\mathcal{T}$

initialize π_0 to any policy (for SSP: proper)
for $i = 0, 1, \dots$:
  compute V_{π_i}
  let π_{i+1} be a greedy policy w.r.t V_{π_i}
  if $\pi_i = \pi_{i+1}$:
    return π_i

Note: if $\pi_i(s) \in A_{V_{\pi_i}(s)}$ then use $\pi_{i+1}(s) := \pi_i(s)$
(only update the policy where necessary).

Properties

- PI computes optimal policy if policy evaluation is exact
- In practice, PI often requires very few iterations ...
- ... and is much faster than solving an LP

Summary

Summary

- Policy evaluation for an acyclic policy is possible in one sweep over the state space with backward induction
- Iterative policy evaluation applies state-value function iteratively and converges to true state-values
- Greedy actions in evaluated policy allow to improve policy
- Policy iteration alternates policy evaluation and policy improvement
- Policy iteration computes an optimal policy (if policy evaluation is exact)