GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

E016350 - Artificial Intelligence

Lecture 17

Decisions & Action

Markov Decision Processes

Aleksandra Pizurica

Ghent University
Fall 2024

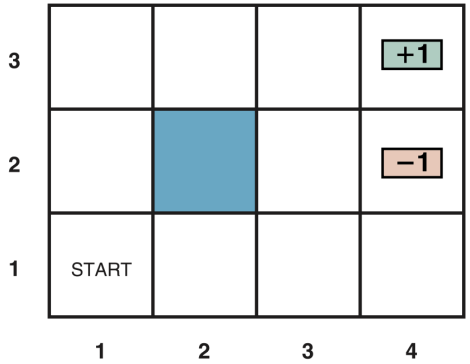
Overview

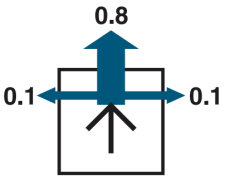
- Making complex decisions
- Markov Decision Process (MDP)
- Value iteration
- Policy iteration
- Alternative formulations of MDPs
  - rewards as functions of a state
  - rewards depending also on the actions and outcomes

[R&N], Chapter 17

These slides are based on: S. Russel and P. Norvig: *Artificial Intelligence: A Modern Approach*, (Fourth Ed.), http://aima.cs.berkeley.edu/, the corresponding slides of S. Russel and the slides of D. Klein and P. Abbeel (course Introduction to Artificial Intelligence), http://ai.berkeley.edu/

A sequential decision problem



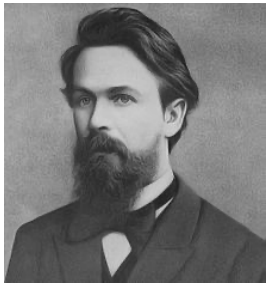


- Noisy actions: e.g., action 'North' takes the agent to North 80% of the time, while 10% of the time takes it East and 10% of the time takes it West.
- When the agent bumps into a wall, it stays put
- A small living reward ($r = -0.04$) in each step. Big rewards $r = \pm 1$ at the end.
- Goal: maximize the sum of rewards.

Markov Decision Process (MDP) – A formal definition

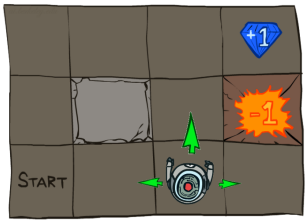
A Markov decision process is defined by:

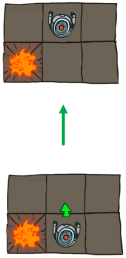
- A set of states $\mathcal{S}$ (with an initial state s_0);
- A set $\mathcal{A}$ of actions in each state: $Actions(s)$
- A transition model $T(s, a, s') = P(s'|s, a)$ and
- A reward function $R(s, a, s')$

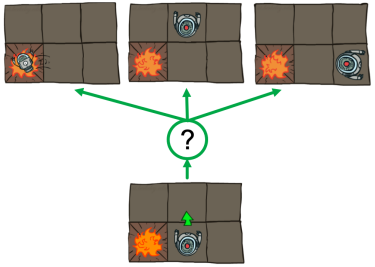


Andrey Markov (1856-1922)

A livelier look at a stochastic grid world



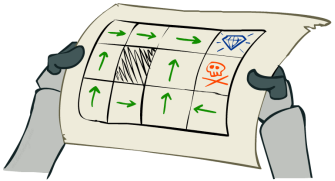




Credit: P. Abbeel and D. Klein, Introduction to Artificial Intelligence

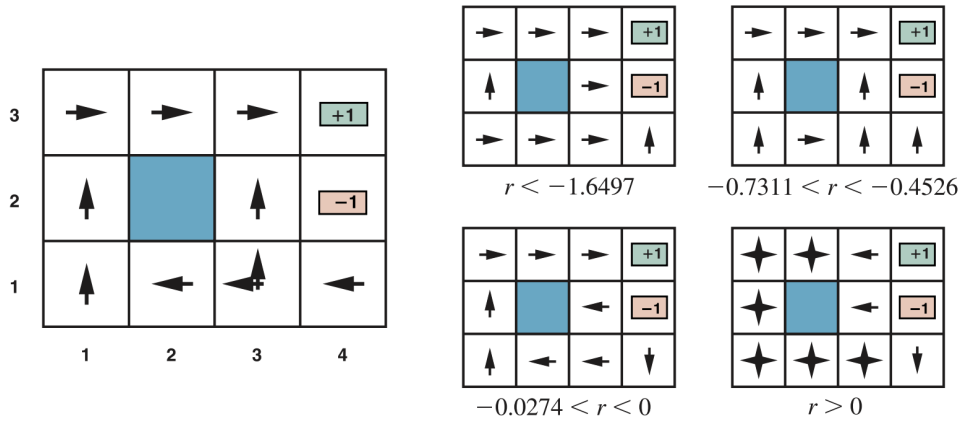
Policies

- In deterministic single-agent search problems, we wanted an optimal plan, or sequence of actions, from start to a goal
- For MDPs, we want optimal policy: $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$
  - A policy gives an action for each state
  - An optimal policy maximizes the expected utility
  - An explicit policy defines a reflex agent
- Expectimax didn't compute entire policies
  - It computed the action for a single state only



P. Abbeel & D. Klein http://ai.berkeley.edu/

Example: optimal policies



Left: Optimal policies for the stochastic environment with $r = -0.04$ for transitions between non-terminal states. **Right**: Optimal policies for different ranges of r.

Utilities over time

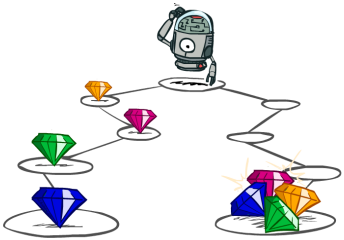
Is there finite horizon or infinite horizon for decision making?

- Let $U_h(s_0, a_0, s_1, a_1, ...s_n)$ be the utility function on environment histories
- Finite horizon: $U_h(s_0, a_0, s_1, a_1, ...s_{N+k}) = U_h(s_0, a_0, s_1, a_1, ...s_N)$
  - May need to consider how much time is left
  - A policy that depends on time is non-stationary
- With no fixed time limit, the optimal policy is stationary
  - Policies for infinite-horizon case are simpler
    but not that simple for partially observable MDP's

Utility of state sequences

What is the utility (value) of the collected rewards?

- The rewards can all come at the end
  or appear on the way
- If they appear on the way, how to combine them?
  - Are they all worth equally?
  - What is the total utility?



P. Abbeel & D. Klein http://ai.berkeley.edu/

Additive discounted rewards

- It is reasonable to maximize the sum of rewards
- Also reasonable to prefer rewards now to rewards later (Why?)
  - One solution: values of rewards decay exponentially
- Leads to additive discounted rewards:

$$U_h(s_0, a_0, s_1, a_1, s_2 ...) = R(s_0, a_0, s_1) + \gamma R(s_1, a_1, s_2) + \gamma^2 R(s_2, a_2, s_3) + ...$$

where $0 \leq \gamma \leq 1$ a discount factor

1

Worth Now

γ

Worth Next Step

γ^2

Worth In Two Steps

Additive infinite rewards

What if the game lasts forever? How to avoid dealing with infinite rewards?
Solutions:

- Finite horizon (similar to depth-limited search)
  - Terminate after k steps
  - Gives non-stationary policies (π depends on the time left)
- Discounting: $0 \leq \gamma \leq 1$
  - Utility of an infinite sequence becomes finite. If the rewards are bounded by $\pm R_{max}$:

$$U_h([s_0, a_0, s_1, ...]) = \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma}$$

  - Smaller γ means a smaller 'horizon' – shorter term focus
- Absorbing state: guarantee that for every policy a terminal state will be reached
  - Proper policy – a policy that is guaranteed to reach a terminal state

Optimal policies and utilities of states

- A policy is a mapping from states to actions: $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$
- A utility (value) function for a policy $U^\pi : \mathcal{S} \rightarrow \mathbb{R}$ gives the expected sum of discounted rewards when acting under that policy. This is the value of the expected utility obtained by executing π starting in s:

$$U^\pi(s) = E\Big[\sum_{t=0}^{\infty} \gamma^t R(S_t, \pi(S_t), S_{t+1})\Big]$$

The optimal policy – one that maximizes the expected utility (starting from s):

$$\pi_s^* = \arg\max_\pi U^\pi(s)$$

and its value is written as $U^* = U^{\pi^*}$

- Note that it in the MDP literature the notation V^π is often used instead of U^π

The Bellman equation

The utility function $U(s)$ allows the agent to select actions by using the principle of maximum expected utility:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$

The utility of being in a state

= the expected sum of discounted rewards from that point onwards

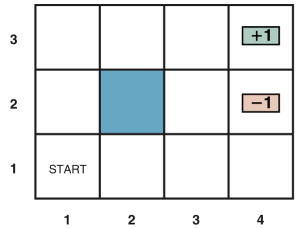
$\Rightarrow \exists$ a direct relationship between

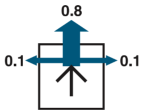
the utility of a state and the utility of its neighbors (Bellman equation):

$$U(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$

The utility of a state is the expected reward for the next transition plus the discounted utility of the next state, assuming that the agent chooses the optimal action.

The Bellman equation: Example





State utilities for $\gamma = 1$ and $r = -0.04$

3	0.8516	0.9078	0.9578	+1
2	0.8016		0.7003	−1
1	0.7453	0.6953	0.6514	0.4279
	1	2	3	4

The Bellman equation for $U(1,1)$, assuming $r = -0.04$:

$$\begin{aligned}\max\{ \quad & [\ 0.8(-0.04 + \gamma U(1,2)) + 0.1(-0.04 + \gamma U(2,1)) + 0.1(-0.04 + \gamma U(1,1)] \\ & [\ 0.9(-0.04 + \gamma U(1,1)) + 0.1(-0.04 + \gamma U(1,2))] \\ & [\ 0.9(-0.04 + \gamma U(1,1)) + 0.1(-0.04 + \gamma U(2,1))] \\ & [\ 0.8(-0.04 + \gamma U(2,1)) + 0.1(-0.04 + \gamma U(1,2)) + 0.1(-0.04 + \gamma U(1,1))] \\ \}\end{aligned}$$

Q-function

- The action-utility function or the Q-function $Q(s, a)$
  = the expected utility of taking a given action in a given state
- The utilities are conveniently expressed in terms of the Q-function:

$$U(s) = \max_a Q(s, a)$$

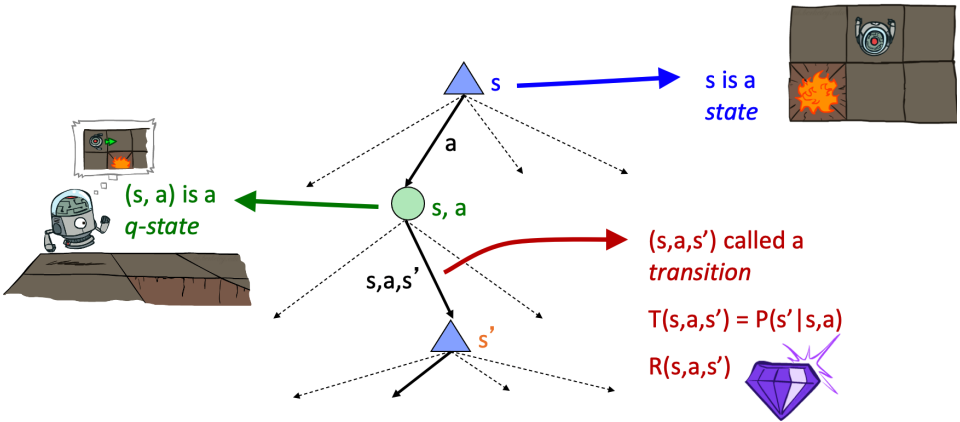
- The optimal policy can be extracted from the Q-function as follows

$$\pi^*(s) = \arg\max_a Q(s, a)$$

- A Bellman equation for Q-functions:

$$\begin{aligned} Q(s, a) &= \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U(s')] \\ &= \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma \max_{a'} Q(s', a')] \end{aligned}$$

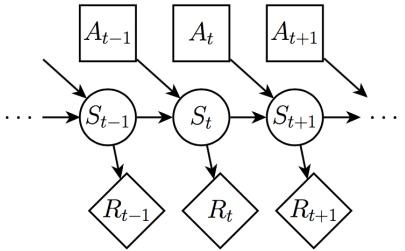
MDP states and expectimax-like search tree

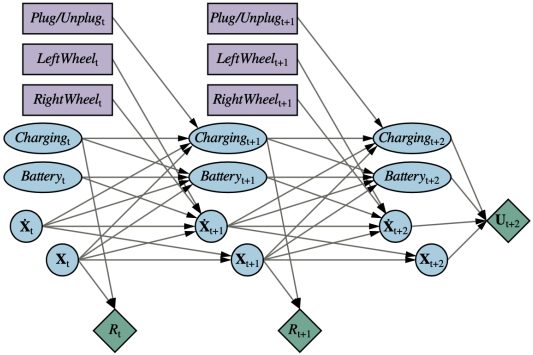


P. Abbeel & D. Klein http://ai.berkeley.edu/

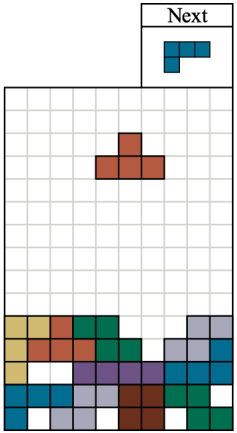
Representing MDPs

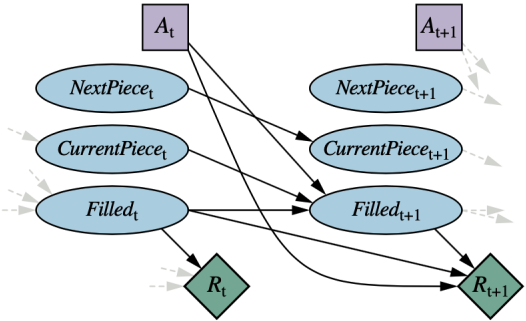
- $P(s'|s,a)$ and $R(s,a,s')$ are 3-dim tables of size $|S|^2|A|$
  for the 4×3 grid world: $11^2 \times 4$ each
- In some cases, tables are sparse, but for large problems even $O(|S||A|)$ is too big
- We represent MDPs by extending Dynamic Bayesian Networks (DBN) with action (decision), reward and utility nodes to create Dynamic Decision Networks (DDN)





Example: Tetris





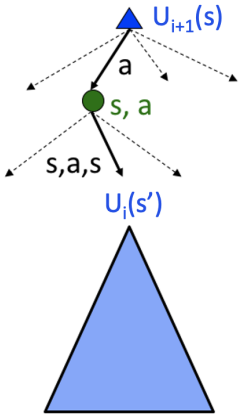
State vars: $CurrentPiece$, $NextPiece$, $Filled$ (bit-vector with one bit per location)
10×20 board locations; State space size: $7 \times 7 \times 2^{200} \approx 10^{62}$

Algorithms for MDPs

- Algorithms for generating exact solutions offline
  - Value iteration
  - Policy iteration
  - Linear programming
- Online approximate algorithms
  - Monte Carlo planning

Value iteration

Iterative approach to solving n nonlinear Bellman equations



1. Initialize $U_0(s) \leftarrow 0$, $\forall s \in \mathcal{S}$
2. Repeat $\forall s$ until convergence:

$$U_{i+1}(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U_i(s')]$$

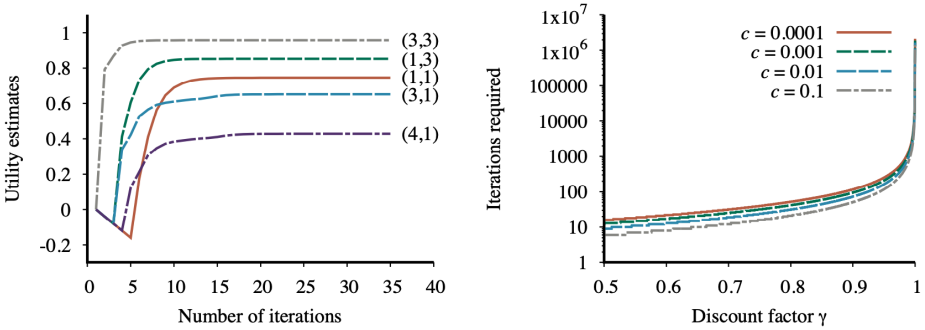
The update is assumed to be applied simultaneously to all the states in each iteration
Converges to a unique set of solutions of the Bellman equations

Value iteration algorithm

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s, a, s')$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum relative change in the utility of any state

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow \max_{a \in A(s)}$ Q-VALUE(mdp, s, a, U)
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta \leq \epsilon(1-\gamma)/\gamma$
  return U

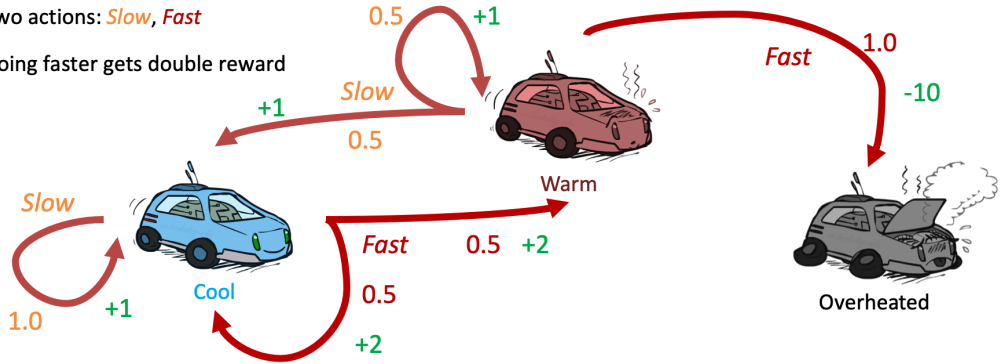
The convergence rate of value iteration: example



Left: Evolution of the utilities of selected states in 4×3 world using value iteration.
Right: The number of value iterations required to guarantee an error of at most $\epsilon = cR_{max}$, for different values of c, as a function γ

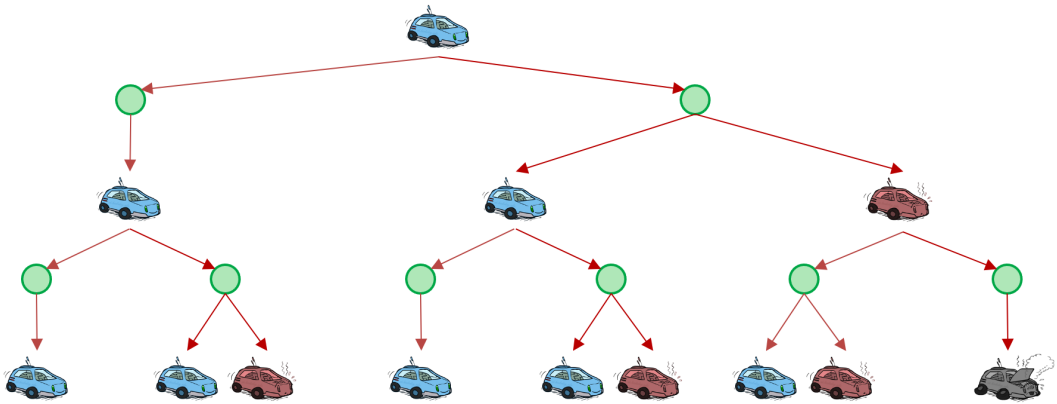
Example: Racing

- A robot car wants to travel far, quickly
- Three states: Cool, Warm, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



Introduction to Artificial Intelligence of P. Abbeel and D. Klein http://ai.berkeley.edu/

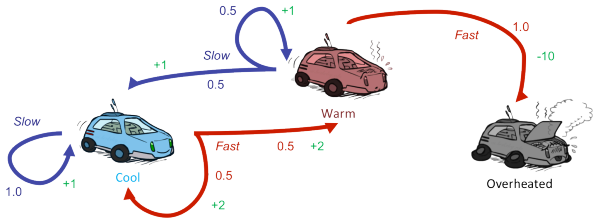
Racing Search Tree



Introduction to Artificial Intelligence of P. Abbeel and D. Klein http://ai.berkeley.edu/

Racing Search Tree

U_2	3.5	2.5	0
U_1	2	1	0
U_0	0	0	0

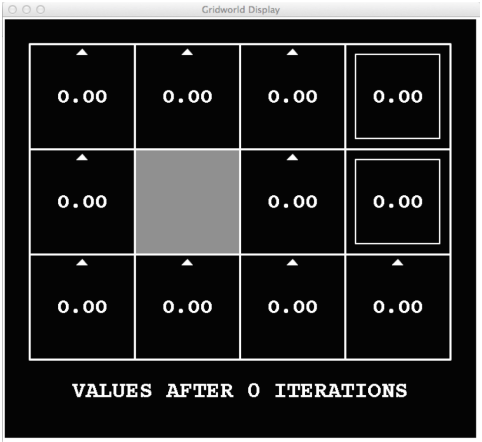


Assume no discount!

$$U_{i+1}(s) = \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U_i(s')]$$

Example: Grid world

$i = 0$

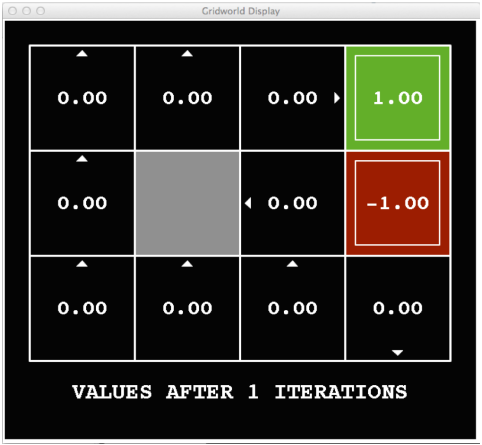


Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

$i = 1$



Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

$i = 2$

Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

$i = 3$

Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

$i = 4$

Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

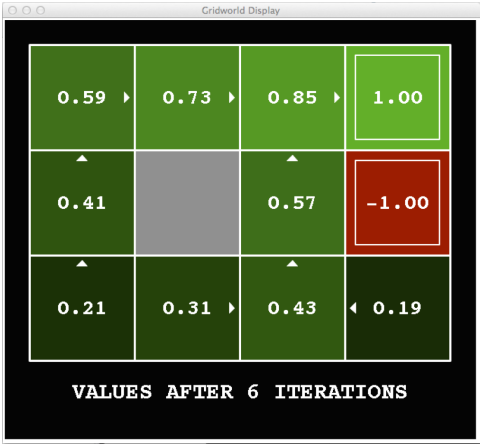
$i = 5$

Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

$i = 6$

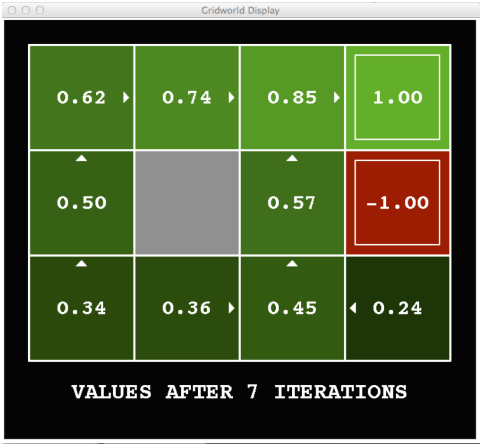


Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

$i = 7$



Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

$i = 8$

Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

$i = 9$

Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

$$i = 10$$

Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

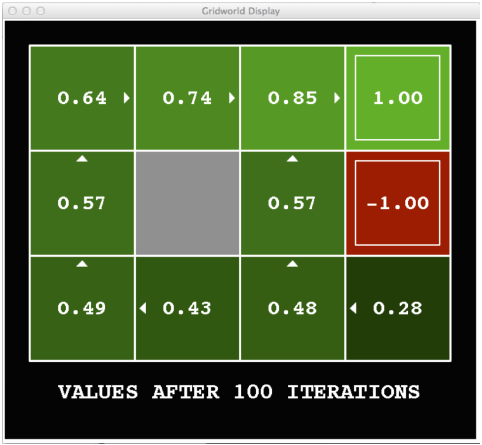
$i = 11$

Noise $= 0.2$; Discount $= 0.9$; Living reward $= 0$

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Example: Grid world

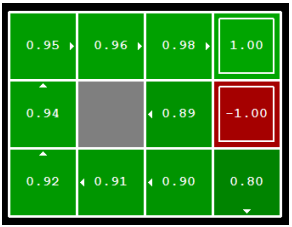
$i = 100$

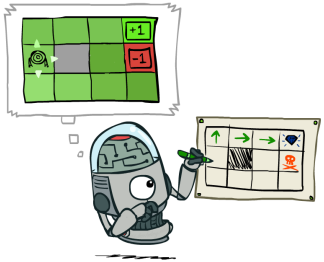


Noise = 0.2; Discount = 0.9; Living reward = 0

P. Abbeel and D. Klein: Introduction to Artificial Intelligence http://ai.berkeley.edu/

Policy extraction





Suppose we computed the optimal values.

How to compute **actions from these values**?

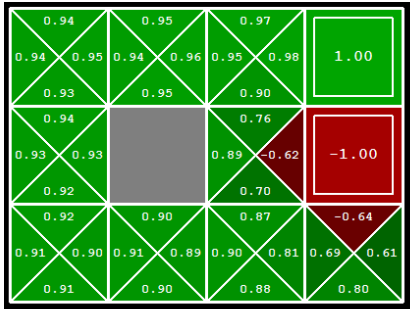
Recall, we already explained:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$$

(one step of expectimax)

Computing actions from Q-Values

Suppose we computed Q-values



How to compute **actions from Q-values**? Easy:

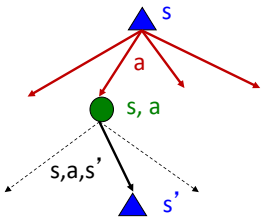
$$\pi^*(s) = \arg\max_{a \in A(s)} Q(s, a)$$

Actions are easier to select from Q-values!

Problems with Value Iteration

Value iteration repeats the Bellman updates:

$$U_{i+1}(s) \leftarrow \max_{a \in A(s)} \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma U_i(s')]$$



Three main problems:

- Slow convergence
- The "max" at each state rarely changes
- The policy often converges long before the values

What the agent really cares about?

– Not the state values but how well it will do if it makes its decisions based on them (in practice it often happens that π_i becomes optimal long before U_i converges)

Policy Iteration

The **policy iteration** algorithm starts from some policy π^0 and alternates two steps:

- Policy evaluation: given a policy π_i, calculate $U_i = U^{\pi_i}$,
  the utility of each state if π_i were to be executed:

$$U_i(s) \leftarrow \sum_{s'} P(s'|s, \pi_i(s))[R(s, \pi_i(s), s') + \gamma U_i(s')]$$

- Policy improvement: Calculate a new MEU policy π_{i+1},
  using one-step look-ahead based on U_i:

$$\pi_{i+1}(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s'} P(s'|s, a)[R(s, a, s') + \gamma U_i(s')]$$

Policy iteration algorithm

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      $a^* \leftarrow \underset{a \in A(s)}{\text{argmax}}$ Q-VALUE(mdp, s, a, U)
      if Q-VALUE(mdp, s, a^*, U) $>$ Q-VALUE($mdp, s, \pi[s], U$) **then**
        $\pi[s] \leftarrow a^*$; $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

Comparison

- Value iteration and policy iteration compute the same thing (all optimal values)
- In value iteration:
  - Every iteration updates both the values and (implicitly) the policy
  - We don't track the policy, but taking the max over actions implicitly recomputes it
- In policy iteration:
  - We do several passes that update utilities with fixed policy
    (each pass is fast because we consider only one action, not all of them)
  - After the policy is evaluated, a new policy is chosen
    (slow like a value iteration pass)
  - The new policy will be better (or we're done)
- Both are dynamic programs for solving MDPs

Partially observable MDPs

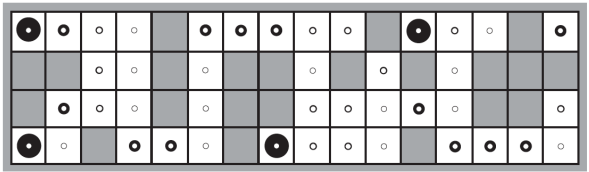
- We assumed so far the environment is fully observable
  the agent knows which state it is in
- If the agent doesn't know which state it is,
  it cannot execute the action $\pi(s)$ recommended for that state

Partially observable MDPs (POMDP)

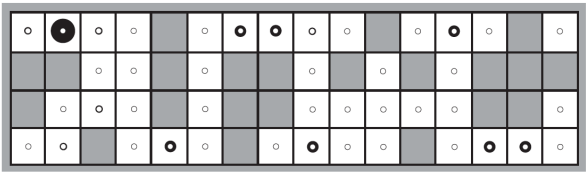
- Has the same elements as MDP (transition model, actions, reward function) and in addition it also has a **sensor model** $P(e|s)$
- The agent computes its belief state

$$b'(s') = \alpha P(e|s') \sum_{s} P(s'|s,a)b(s)$$

Partially observable MDPs



(a) Posterior distribution over robot location after E_1 = NSW



(b) Posterior distribution over robot location after E_1 = NSW, E_2 = NS

Remember the lesson on temporal probability models

The agent computes the belief state

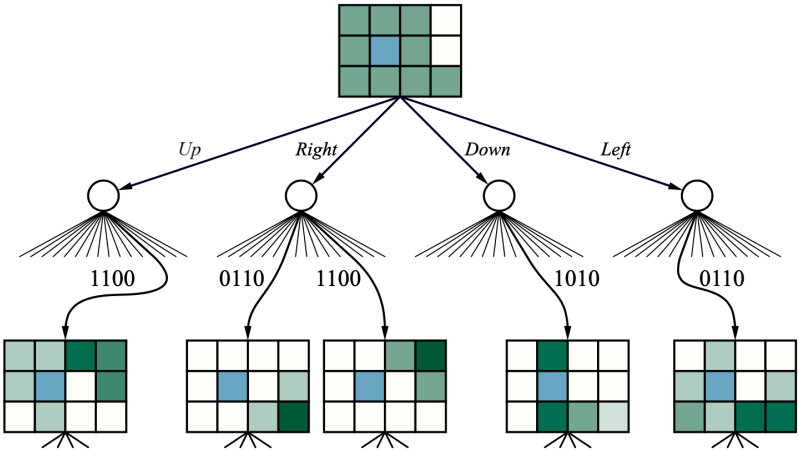
$$b'(s') = \alpha P(e|s') \sum_s P(s'|s,a)b(s)$$

This is essentially the **filtering** task

$$b' = \textsc{Forward}(b, a, e)$$

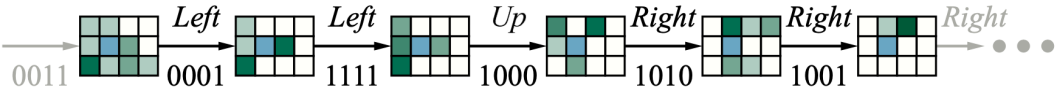
Posterior distribution $P(X_t = i|\mathbf{e}_t)$ over robot location: (a) one observation $E_1 = NSW = 1011$; (b) after a second observation $E_2 = NS = 1010$. The size of each disk corresponds to the probability that the robot is at that location. $\epsilon = 0.2$

Partially observable MDPs: Example 4×3 world



Part of an expectimax tree for the 4×3 POMDP with a uniform initial belief state. Belief states depicted with shading $\propto$ the probability of being in each location.

Partially observable MDPs: Example 4×3 world



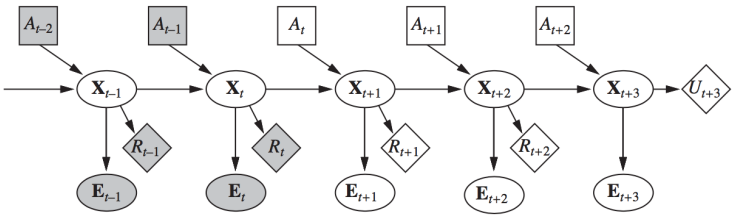
The decision circle of a POMPD agent can be broken into 3 steps:

1. Given the current belief state b, execute the action $a = \pi^*(b)$
2. Observe the percept e
3. Set the current belief state to $b' = \text{FORWARD}(b, a, e)$ and repeat

The probability of perceiving e, given that a was performed starting in belief state b:

$$\begin{aligned} P(e|a,b) &= \sum_{s'} P(e|a,s',b)P(s'|a,b) \\ &= \sum_{s'} P(e|s')P(s'|a,b) = \sum_{s'} P(e|s') \sum_{s} P(s'|s,a)b(s) \end{aligned}$$

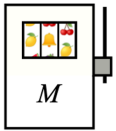
POMDP represented as Dynamic Decision Networks



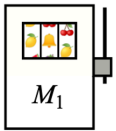
- Variables with known values are shaded
- The current time is t and the agent must choose a value for the action A_t
- The network has been unrolled into the future for three steps and represents future rewards, as well as the utility of the state at the look-ahead horizon

(Source: R&N, 3rd Ed)

Bandit Problems



0, 2, 0, 7.2, 0, 0, 0, …



1, 1, 1, 1, 1, 1, 1, …

Suppose

- $\gamma = 0.5$
- two arms: M and M_1

Choose which one to play

$$U(M) = (1 \times 0) + (0.5 \times 2) + (0.5^2 \times 0) + (0.5^3 \times 7.2) = 1.9$$

$$U(M_1) = \sum_{t=0}^{\infty} 0.5^t = 2$$

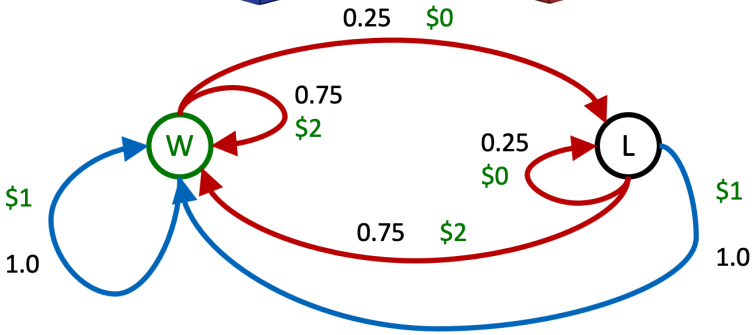
How about switching? E.g., after fourth reward:

$$U(S) = (1 \times 0) + (0.5 \times 2) + (0.5^2 \times 0) + (0.5^3 \times 7.2) + \sum_{t=4}^{\infty} 0.5^t = 2.025$$

Bandit Problems

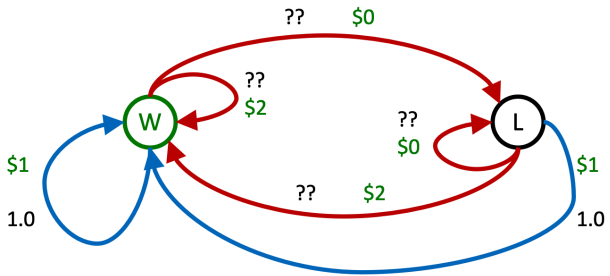
Actions: *Blue*, *Red*

States: Win, Lose



Online planning

Rules changed! Red's win chance is different.



$0 $0 $0 $2 $0
$2 $0 $0 $0 $0

Has to involve exploration. This will be a reinforcement learning problem!

Summary

- MDPs are sequential decision problems in uncertain environments.
- The MDPs are defined by a transition model, specifying the probabilistic outcomes of actions and by a reward function specifying a reward for each state (and possibly depending on the action and the outcome as well)
- The solution to an MDP is a policy
- We analysed two approaches for computing optimal policies: value iteration and policy iteration.
- Partially observable MDPs, called POMDPs are much more difficult to solve. Involve computing belief states (filtering operation) and can be represented by Dynamic Bayesian networks
- MDPs and POMDPs where the agent improves its performance based on experience $\rightarrow$ reinforcement learning