P11.Markov Decision Processes

Radek Mařík

CVUT FEL, K13133

22. dubna 2013

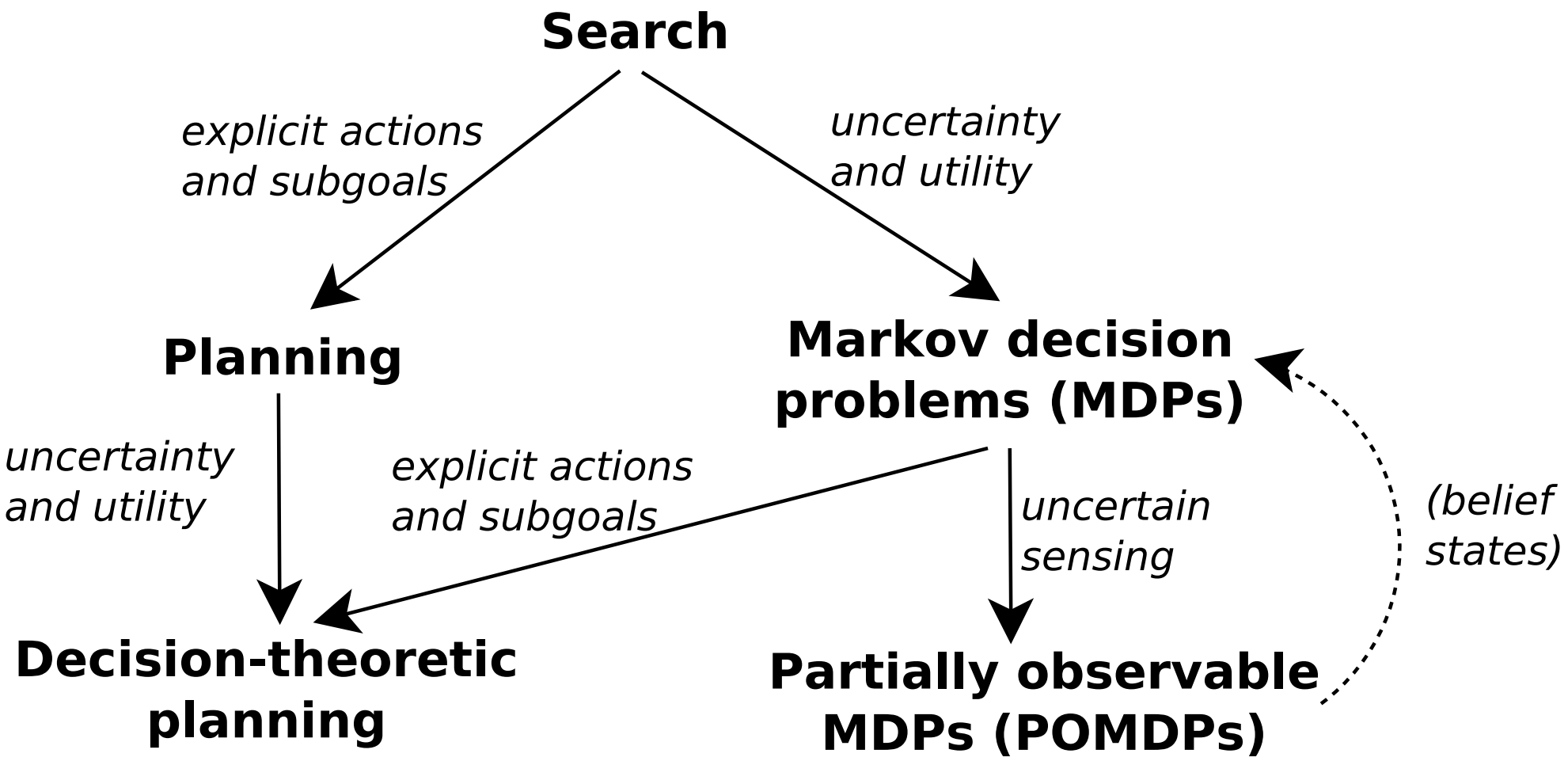
Obsah

1. Introduction
   - Introduction
2. Markov Decision Process
   - Markov Decision Process
   - Utility Function, Policy
3. Solving MDPs
   - Value Iteration
   - Policy Iteration
4. Conclusions
   - Conclusions

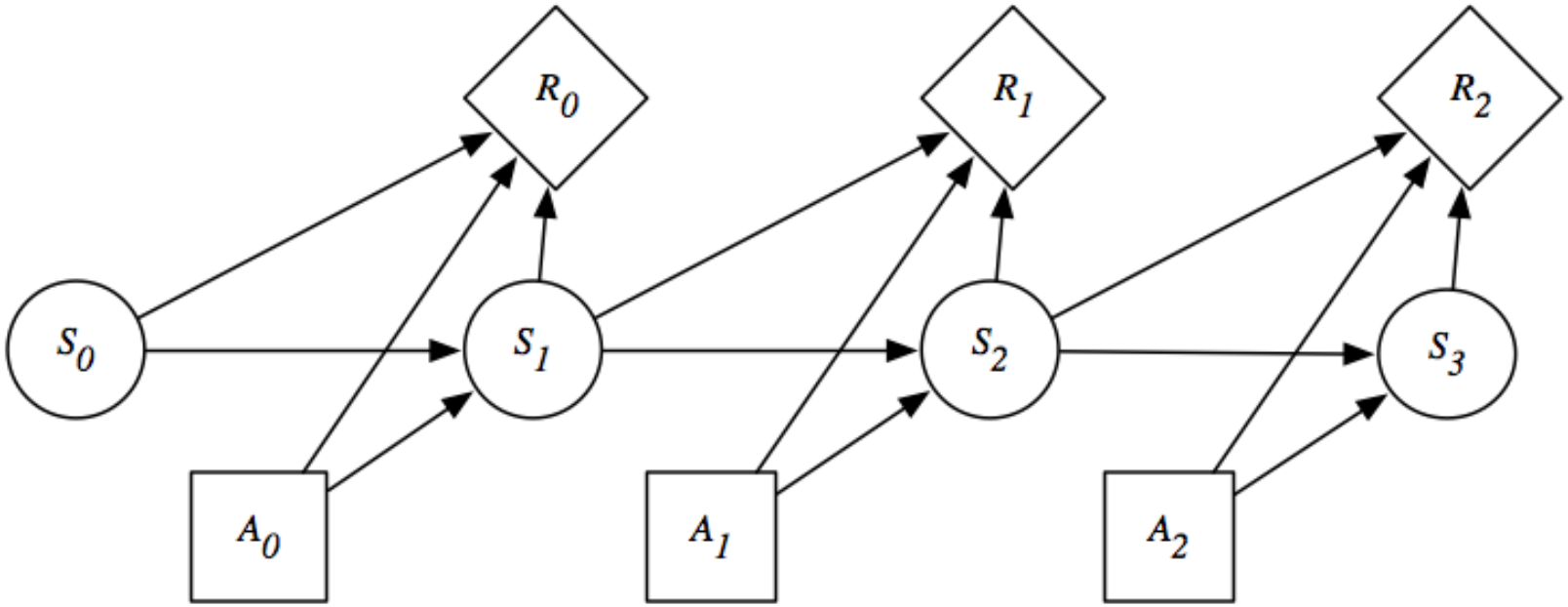
Sequential Decisions [RN10]

- Achieving agent's objectives often requires multiple steps.
- A rational agent does not make a multi-step decision and carry it out without considering revising it based on future information.
  - Subsequent actions can depend on what is observed
  - What is observed depends on previous actions
- Agent wants to maximize reward accumulated along its course of action
- What should the agent do if environment is non-deterministic?
  - Classical planning will not work
  - Focus on state sequences instead of action sequences

Sequential Decision Problems [Jak10]



Markov Decision Process [PM10]



Markov Decision Process [PM10]

Definition (Markov Decision Process)

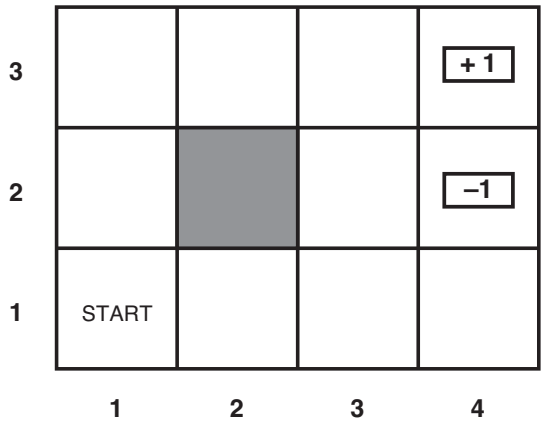
A **Markov Decision Process (MDP)** is a 5-tuple $\langle S, A, T, R, s_0 \rangle$ where

- S is a set of states
- A is a set of actions
- $T(S, A, S')$ is the transition model
- $R(S)$ is the reward function
- s_0 is the initial state

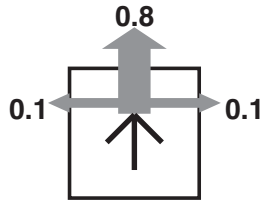
- Transitions are Markovian

$$P(S_n|A, S_{n-1}) = P(S_n|A, S_{n-1}, S_{n-2}, \ldots, S_0) = T(S_{n-1}, A, S_n)$$

Example: Simple Grid World [RN10]



(a)



(b)

Simple 4x3 environment

- States $S = \{(i,j) | 1 \leq i \leq 4 \wedge 1 \leq j \leq 3\}$
- Actions $A = \{up, down, left, rigth\}$
- Reward function

$$R(s) = \begin{cases} -0.04 & \text{(small penalty) for nonterminal states} \\ \pm 1 & \text{for terminal states} \end{cases}$$

- Transition model $T((i,j), a, (i',j'))$ given by (b)

Utility Function [RN10, Jak10]

- Utility function captures agent's preferences
  - In sequential decison-making, utility is a function over sequences of states
- Utility function accumulates rewards:
  - Additive rewards (special case):

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + R(s_1) + R(s_2) + \dots$$

  - Discounted rewards

$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

    where $\gamma \in [0, 1]$ is the discount factor
- Discounted rewards for $\gamma < 1$ finite even for infinite horizons (see next slide)
- No other way of assigning utilities to state sequences is possible assuming stationary preferences between state sequences

Policy [RN10, Jak10]

- A stationary policy is a function

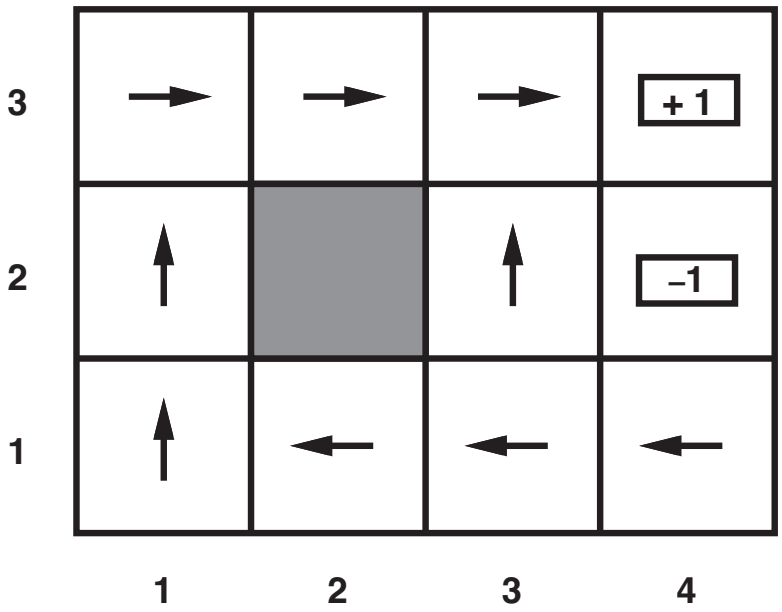
$$\pi : S \to A$$

- Optimal policy is a function maximizing expected utility

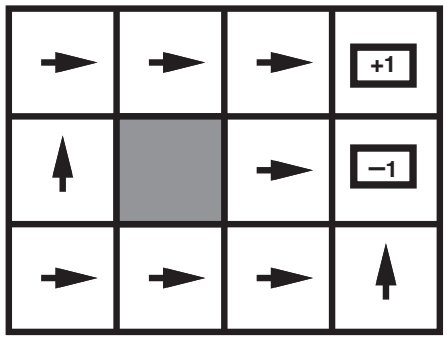
$$\pi^\star = \arg\max_\pi E[U([s_0, s_1, s_2, \dots])|\pi]$$

- For an MDP with stationary dynamics and rewards with infinite horizon, there always exists an optimal stationary policy
  - no benefit to randomize even if environment is random

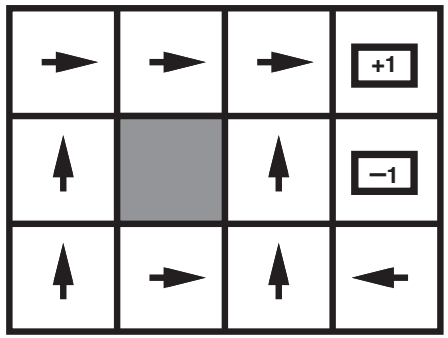
Example: Optimal Policies in the Grid World [RN10, Jak10]



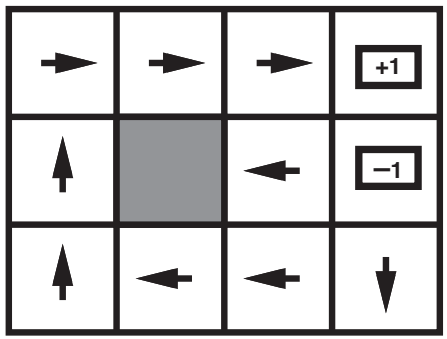
(a)



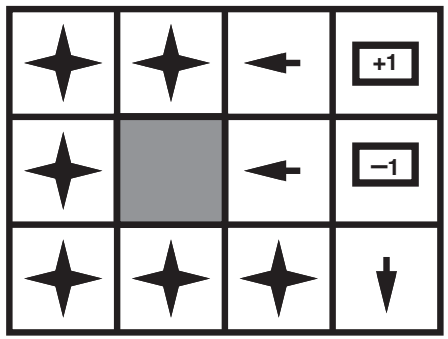
$R(s) < -1.6284$



$-0.4278 < R(s) < -0.0850$



$-0.0221 < R(s) < 0$



$R(s) > 0$

(b)

- (a) Optimal policy for state penalty $R(s) = -0.04$
- (b) Dependence on penalty

Decision-making Horizon [RN10, Jak10]

- A finite horizon means that there is a finite deadline N after which nothing matters (the game is over)
  - $\forall k \geq 1 \quad U_h([s_0, s_1, \ldots, s_{N+k}]) = U_h([s_0, s_1, \ldots, s_N])$
  - The optimal policy is non-stationary, i.e., it could change over time as the deadline approaches.
- An infinite horizon means that there is no deadline
  - The optimal policy is stationary $\Leftarrow$ there is no reason to behave differently in the same state at different times
  - Easier than the finite horizon case
- terminate / absorbing states – agents stay there forever receiving zero reward at each step

Solving MDPs [RN10, Jak10]

- How do we find the optimum policy π^*?
- Two basic techniques:
  1. value iteration – compute utility $U(s)$ for each state and use is for selecting best action
  2. policy iteration – represent policy explicitly and update it in parallel to the utility function

Utility of State [RN10, Jak10]

- Utility of a state under a given policy π:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right]$$

- True utility $U(s)$ of a state is the utility assuming optimum policy π^*

$$U(s) := U^{\pi^*}(s)$$

  - Reward $R(s)$ is "short-term" reward for being in s;
    utility $U(s)$ is a "long-term" total reward from s onwards
- Selecting the optimum action according to
  the MEU (Maximum Expected Utility) principle

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s, a, s') U(s')$$

Bellman Equation [RN10, Jak10]

- Definition of utility of states leads to a simple relationship among utilities of neighboring states
- The utility of a state is the immediate reward for the state plus the expected discounted utility of the next state, assuming the agent chooses the optimal action

Definition (Bellman equation (1957))

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U(s') \quad \forall s \in S$$

- One equation per state $\Rightarrow$ n non-linear equations for n unknowns
  - The solution is unique

Iterative Solution [RN10, Jak10]

- Analytical solution is not possible ⇒ iterative approach

Definition (Bellman update)

$$U_{i+1}(s) = R(s) + \gamma \max_a \sum_{s'} T(s, a, s') U_i(s') \quad \forall s \in S$$

- Dynamic programming: given an estimate of the k-step lookahead value function, determine the $k+1$-step lookahead utility function.
- If applied infinitely often, guaranteed to reach an equilibrium and the final utility values are the solutions to the Bellman equations
- Value iteration propagates information through the state space by means of local updates.

Value Iteration Algorithm [RN10, Jak10]

Input: mdp, a MDP with states S, transition model T, reward function R, discount γ

Input: ϵ, the maximum error allowed in the utility of a state

Local variables: U, U', vectors of utilities for states in S, initially zero

Local variables: δ, the maximum change in the utility of any state in an iteration

repeat

  $U \leftarrow U'; \delta \leftarrow 0$;

  foreach *state* $s \in S$ **do**

    $U'[s] \leftarrow R[s] + \gamma \max_a \sum_{S'} T(s, a, s')U[s']$;

    if $|U'[s] - U[s]| > \delta$ **then**

      $\delta \leftarrow |U'[s] - U[s]|$;

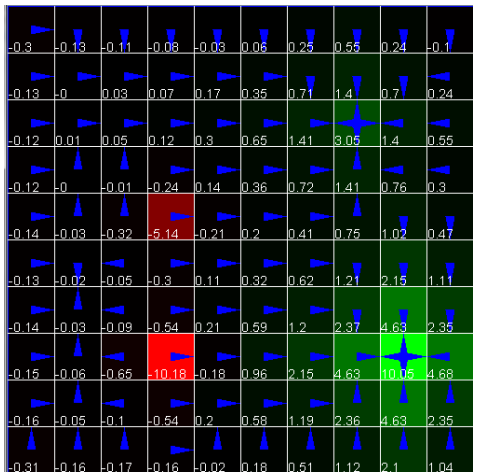
    end

  end

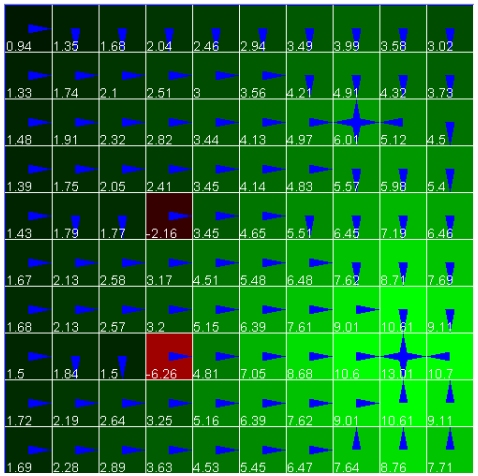
until $\delta < \epsilon(1 - \gamma)/\gamma$;

return U

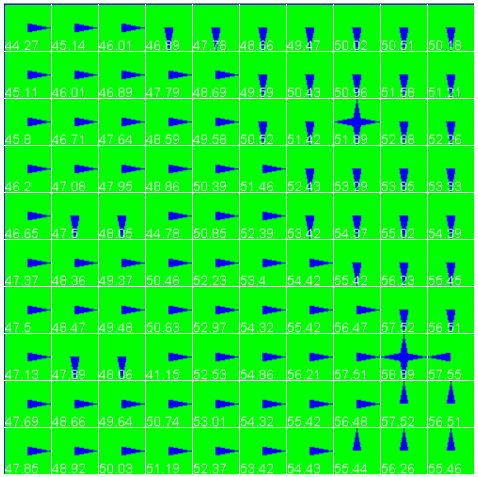
Value Iteration Example [RN10, PM10, Jak10]



(a) $\gamma = 0.6$



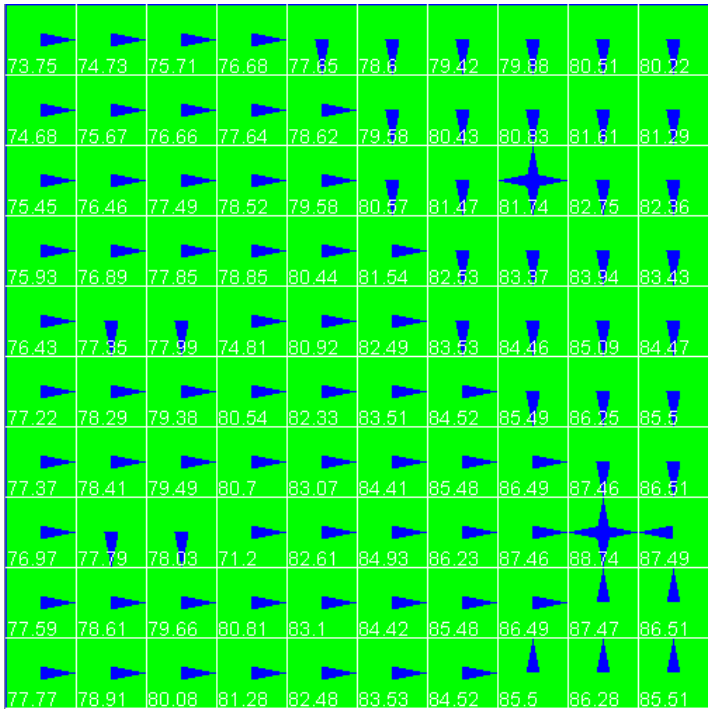
(b) $\gamma = 0.9$



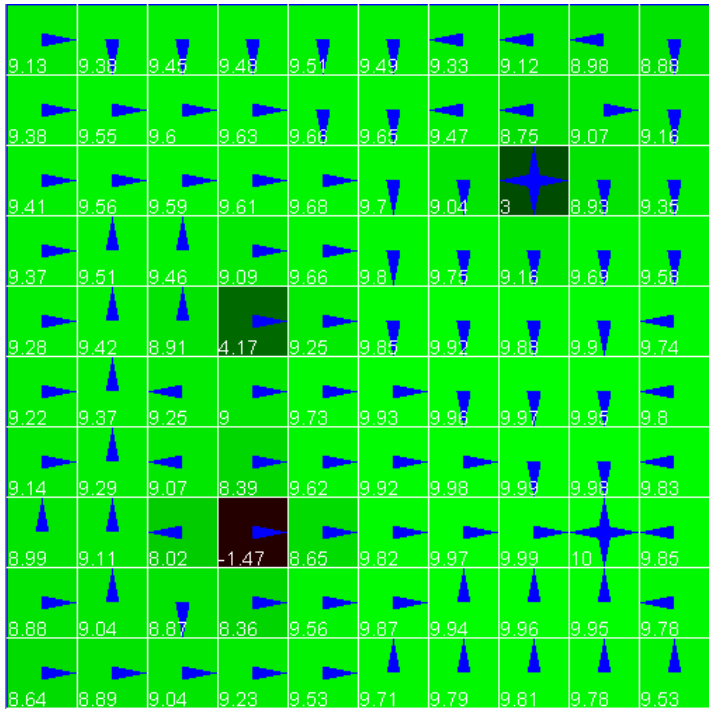
(c) $\gamma = 0.99$

- 4 movement actions; 0.7 chance of moving in the desired direction, 0.1 in the others
- $R = -1$ for bumping into walls; four special rewarding states
  - +10 (at position (9,8); 9 across and 8 down),
  - one worth +3 (at position (8,3)),
  - one worth -5 (at position (4,5)) and
  - one -10 (at position (4,8))

Value Iteration Example [RN10, PM10, Jak10]



(a) $\gamma = 1.0$, non-absorbing



(b) $\gamma = 1.0$, absorbing

- differences in convergence for different γ while absorbing states are present or not

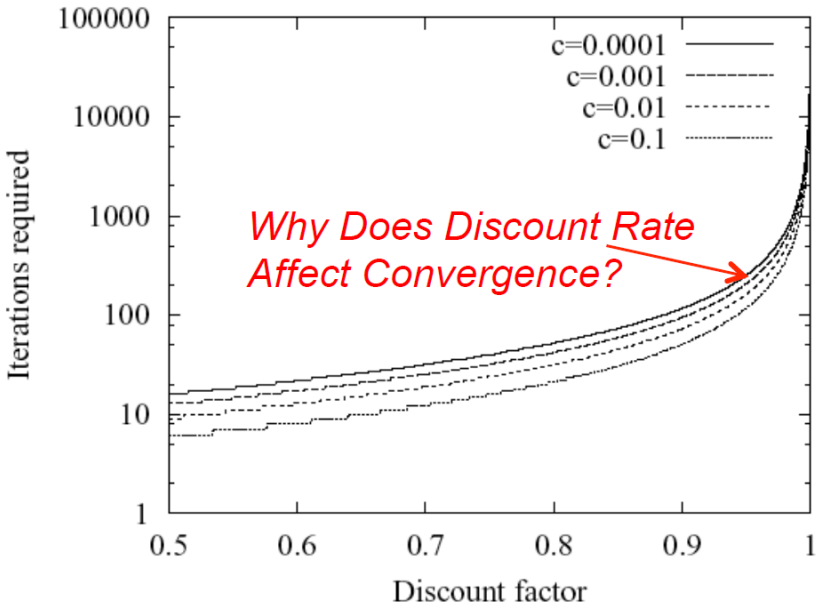
Value Iteration Example [RN10, PM10, Jak10]

- Dependence on discount γ and existence of absorbing states

discount γ	absorbing states	steps to convergence
0.6	no	10
0.9	no	10s
0.99	no	100s
1.0	no	∞ *(no convergence)*
1.0	yes	10s

- See `http://artint.info/demos/mdp/vi.html`

Convergence [RN10, Jak10]



- Value iteration converges to the correct utilities
- We can bound the error in the utility estimates if we stop after a finite number of iterations, and we can bound the policy loss that results from executing the corresponding MEU policy.
- MEU policy may be optimal long before convergence!
  - Optimal policy not very sensitive to the utility values

Policy Iteration [RN10, Jak10]

- Search for optimal policy and utility values simultaneously
- Alternates between two steps:
  1. policy evaluation – recalculates values of states $U_i = U^{\pi_i}$ given the current policy π_i
  2. policy improvement/iteration – calculates a new MEU policy π_{i+1} using one-step look-ahead based on U_i
- Terminates when the policy improvement step yields no change in the utilities.

Policy Iteration Algorithm [RN10, Jak10]

Input: mdp, a MDP with states S, transition model T
Local variables: U, a vector of utilities for states in S, initially zero
Local variables: π, a policy vector indexed by state, initially random
repeat
  $U \leftarrow$ `Policy-Evaluation`(π, U, mdp) ;
  $unchanged? \leftarrow$ true;
  foreach *state* $s \in S$ **do**
    if $\max_a \sum_{S'} T(s,a,s')U[s'] > \sum_{S'} T(s,\pi(s),s')U[s']$ **then**
      $\pi(s) \leftarrow \arg\max_a \sum_{S'} T(s,a,s')U[s']$;
    end
    $unchanged? \leftarrow$ false;
  end
until $unchanged?$;
return π

Policy Evaluation [RN10, Jak10]

- Simplified Bellman equations:

$$U_i(s) = R(s) + \gamma \sum_{S'} T(s, \pi_i(s), s') U_i(s') \quad \forall s \in S$$

- The equations are now linear $\Rightarrow$ can be solved in $O(n^3)$

Modified Policy Iteration [RN10, Jak10]

- Policy iteration often converges in few iterations but each iteration is expensive
  - $\Leftarrow$ has to solve large systems of linear equations
- Main idea: use iterative approximate policy evaluation
  - Simplified Bellman update:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{S'} T(s, \pi_i(s), s') U_i(s') \quad \forall s \in S$$

  - Use a few steps of value iteration (with π fixed)
  - Start from the value function produced in the last iteration
- Often converges much faster than pure value iteration or policy iteration (combines the strength of both approaches)
- Enables much more general asynchronous algorithms
  - e.g. Prioritized sweeping

Choosing the Right Technique [RN10, Jak10]

- Many actions?⇒ policy iteration
- Already got a fair policy? ⇒ policy iteration
- Few actions, acyclic? ⇒ value iteration
- Modified policy iteration typically the best

Conclusions [RN10, Jak10]

- MDPs generalize deterministic state space search to stochastic environments
  - At the expense of computational complexity
- An optimum policy associates an optimal action with every state
- Iterative techniques used to calculate optimum policies
  - basic: value iteration and policy iteration
  - improved: modified policy iteration, asynchronous policy iteration
- Further issues
  - large state spaces – use state space approximation
  - partial observability (POMDPs) – need to consider information gathering; can be mapped to MDPs over continous belief space

Acknowledgement

- The first version of this presentation was prepared as a clone of presentations created by Michal Jakob [Jak10] and David Poole [PM10]

References I

Michal Jakob.

A3M33UI decision theory essentials, lecture notes.
http://cw.felk.cvut.cz/doku.php/courses/a3m33ui/prednasky, February 2010.

David Poole and Alan Mackworth.

Artificial intelligence, foundations of computational agents.
http://artint.info/slides/slides.html, 2010.

Stuart J. Russell and Peter Norvig.

Artificial Intelligence, A Modern Approach.
Pre, third edition, 2010.