Sequential decision making under uncertainty

Jiří Kléma

Department of Computer Science,
Czech Technical University in Prague

https://cw.fel.cvut.cz/wiki/courses/b4b36zui/prednasky

Agenda

- Previous lecture: individual rational decisions under uncertainty
  - uncertainty = stochastic action outcomes – lottery, expected utility,
- how to optimally choose whole sequences of actions?
  - repeated decisions based on uncertain or incomplete information,
  - prizes/rewards typically delayed,
  - world does not have to be fully observable,
- Markov decision process
  - introduces Markov assumption/property – process with a limited memory,
  - works with stationarity and observability assumptions too,
  - world/environment well defined by its transition and reward functions,
- generalization in the next lecture
  - POMDP – the world is partially observable only,
  - reinforcement learning – no environment model available, no state-transition and reward functions.

Markov process

- random process, prob of visiting future states given by recent states only,
- distant past is irrelevant provided that we know the recent past,
- **Markov** chain
  - discrete random process with Markov property,
  - chain **order** m gives how many past states we need to concern

    $$P(X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) = P(X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_{n-m} = x_{n-m})$$
  - most often 1st order models,
  - commonly together with stationarity assumption (time invariance)

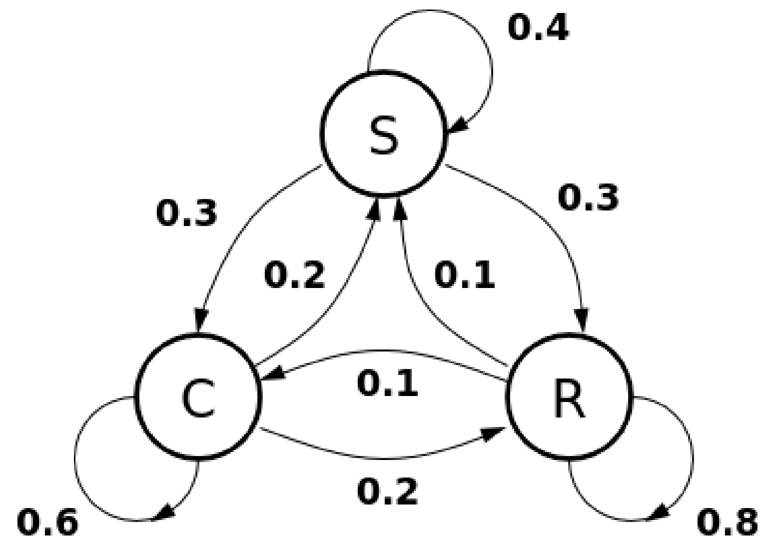
    $$Pr(X_{n+1} = x | X_n = y) = Pr(X_n = x | X_{n-1} = y)$$
- examples of Markov chains
  - coin tosses – HTHHHT ...
    - degenerate (zero order) Markov chain,
  - weather observed every day at noon – SSSCRRCSRR ...
    - categorized (S)unny, (C)loudy, (R)ain, the order is unknown.

Markov chain – weather example

- How to build a model based on the observed sequence?
  - let us have a sequence of length 41, assume 1st order,
    SSCCRRRRRRRRSSSCRRRRRRRCCCSCSSRRSRRRCSCCC
  - model = transition matrix.

$$
\begin{array}{c} \\ S_t \\ C_t \\ R_t \end{array}
\begin{array}{c} S_{t+1} C_{t+1} R_{t+1} \\ \begin{bmatrix} 4 & 3 & 3 \\ 2 & 6 & 2 \\ 2 & 2 & 16 \end{bmatrix} \end{array}
$$

$$
A = \{a_{ij}\} = \begin{bmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{bmatrix}
$$



Markov chain – weather example

- Which questions can be answered with the Markov model of the sequence?
  1. sunny today, probability that weather will be SSRRSCS for next 7 days?
  2. model in a known state, prob that the state will not change for d days?
  3. what is the expected value of d in the individual states?

Markov chain – weather example

- Which questions can be answered with the Markov model of the sequence?
  1. sunny today, probability that weather will be SSRRSCS for next 7 days?
  2. model in a known state, prob that the state will not change for d days?
  3. what is the expected value of d in the individual states?

- Solution

1. $P(O|M) = P(S,S,S,R,R,S,C,S|M) =$
$= P(S)\ P(S|S)\ P(S|S)\ P(R|S)\ P(R|R)\ P(S|R)\ P(C|S)\ P(S|C) =$
$= 1 \times 0.4 \times 0.4 \times 0.3 \times 0.8 \times 0.1 \times 0.3 \times 0.2 = 2.3 \times 10^{-4}$

2. $O = \{\underbrace{Q_i, Q_i, \ldots, Q_i}_{d}, Q_j \neq Q_i\}$, $P(O|M, q_1 = Q_i) = a_{ii}^{d-1}(1 - a_{ii}) = p_i(d)$,

3. $\bar{d}_i = \sum_{d=1}^{\infty} d p_i(d) = \sum_{d=1}^{\infty} d a_{ii}^{d-1}(1 - a_{ii}) = \frac{1}{1-a_{ii}}$,
(sum of arithmetic-geometric series: $\sum_{k=0}^{\infty} k r^{k-1} = \frac{1}{(1-r)^2}$),
$d_S = 1.67$, $d_C = 2.5$, $d_R = 5$.

Sequential decision making under uncertainty

- commonly there are more steps/actions needed to reach the goal,
- let us assume
  - non-deterministic environment (actions with uncertain outcomes),
  - the goal state replaced by the aim of maximizing cumulative **reward**,
- the sequence of actions cannot be found by classical planning
  - rational agent **re-examines** its steps during the process of solution (execution of actions),
  - next action depends on current observations,
  - current observations depend on current state (= previous actions),
- solution
  - agent evaluates states instead of direct creation of action sequences,
  - in each state take the action leading to successor states with highest value.

Basic concepts, problem definition

- Reward R_t
  - simple sum of immediate rewards obtained per episode:
    $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T$
  - discounted sum for infinite processes
    (γ is discount rate, $0 \leq \gamma \leq 1$):
    $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$
- Policy $\pi_t(s, a)$
  - is a mapping between states and actions,
  - it gives probability that action a will be executed in state s,
  - optimal policy π^* maximizes the total reward R_t,

Basic concepts, problem definition

- State value $V^\pi(s)$
  - expected (cumulative) reward for following policy π starting from state s

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- Action value $Q^\pi(s,a)$
  - expected (cumulative) reward starting from state s, taking action a and thereafter following π

$$Q^\pi(s,a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

- Goal: find π^* (altogether with V^*, Q^* that serve as means).

Sequential decision making as finite MDP

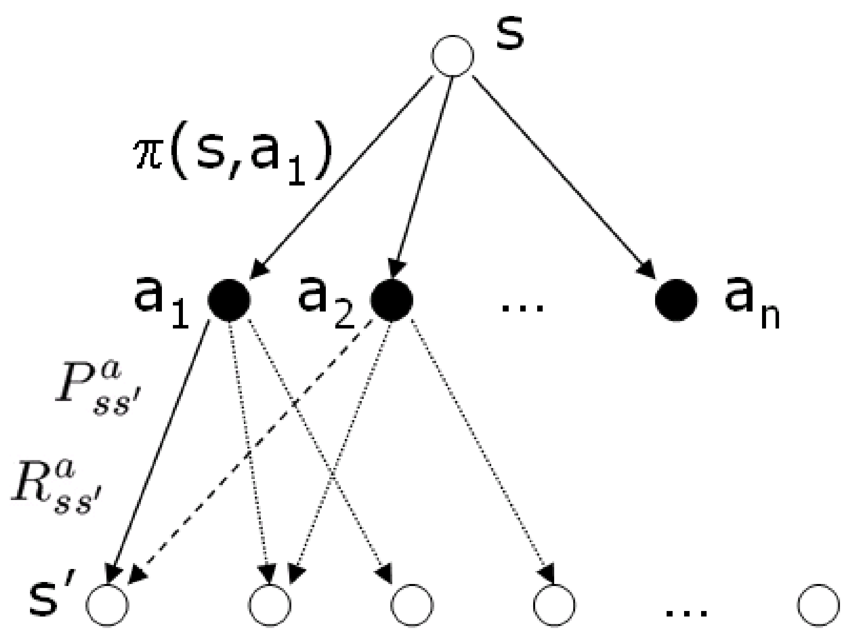
- Finite Markov Decision Process (MDP)
  - Markov assumption + the sets of states S and actions A are finite,
  - MDP = {S,A,P,R}, can be written as a transition graph,
  - P transition probability, R reward function,

$$P^a_{ss'} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$
$$R^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

  - this definition leads to particular values of V and Q,
- implicit assumptions
  - environment is observable (the current state is always known),
  - environment is describable (P and R known),
  - counter example: blackjack card game (reaching P and R a part of solution).

Sequential decision making as finite MDP

- How to obtain state values from a known environment and policy?
  - by transition to the recursive V definition,
  - state value = immediate reward for action execution + expected reward for development of possible successor states.



Sutton, Barto: Reinforcement Learning: An Introduction.

Recursive V definition (Bellman equation)

$$
\begin{aligned}
V^\pi(s) &= E_\pi\{R_t \mid s_t = s\} = \\
&= E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} = \\
&= E_\pi\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\right\} = \\
&= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_{t+1} = s'\right\}\right] = \\
&= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s')\right]
\end{aligned}
$$

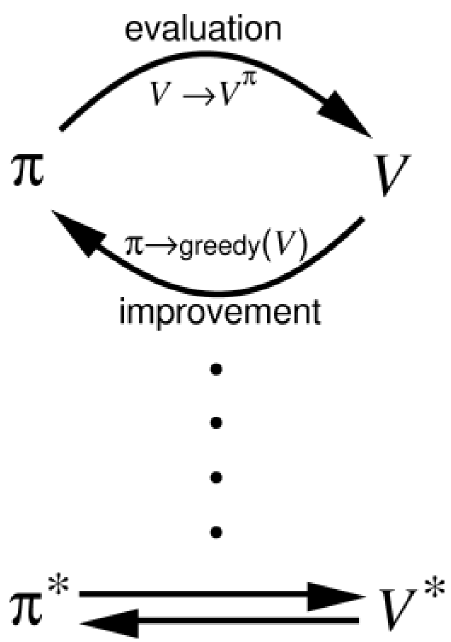
- in the beginning $V^\pi(s)$, $V^\pi(s')$ and $\pi(s,a)$ unknown
  - iterative calculation/improvement,
  - **bootstrapping** – effort analogous to one who would lift himself by his own bootstraps.

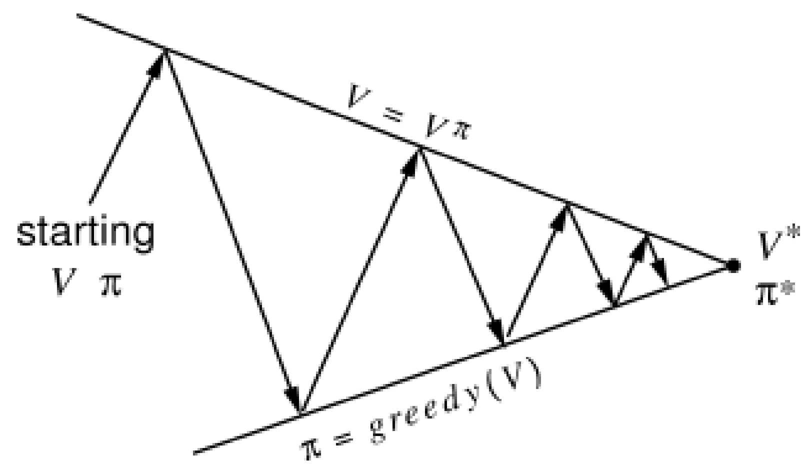
Dynamic programming

- The basic approach to solve MDP (find π^*)
  - dynamic = iterative procedure
    - to find $V(s)$ in step $k+1$ use $V(s')$ from step k,
  - programming = searching for an acceptable sequence of actions,
- polynomial complexity in the number of states $|S|$ and actions $|A|$
  - despite the space of policies with cardinality $|A|^{|S|}$,
  - state space search necessarily performs worse,
  - still often intractable for real problems
    - the above-mentioned estimate holds for single iteration,
    - the number of iterations can be large (exponential when $\gamma \to 1$),
    - unknown process parameters (see reinforcement learning),
    - computationally intractable
      - often too many states,
      - we cannot iterate systematically – asynchronous DP.

Dependency between policy and value functions

- When solving MDP, one simultaneously and interactively
  - adapts state/action values according to the current policy,
  - adapts the policy to maximize reward given the current state/action values.



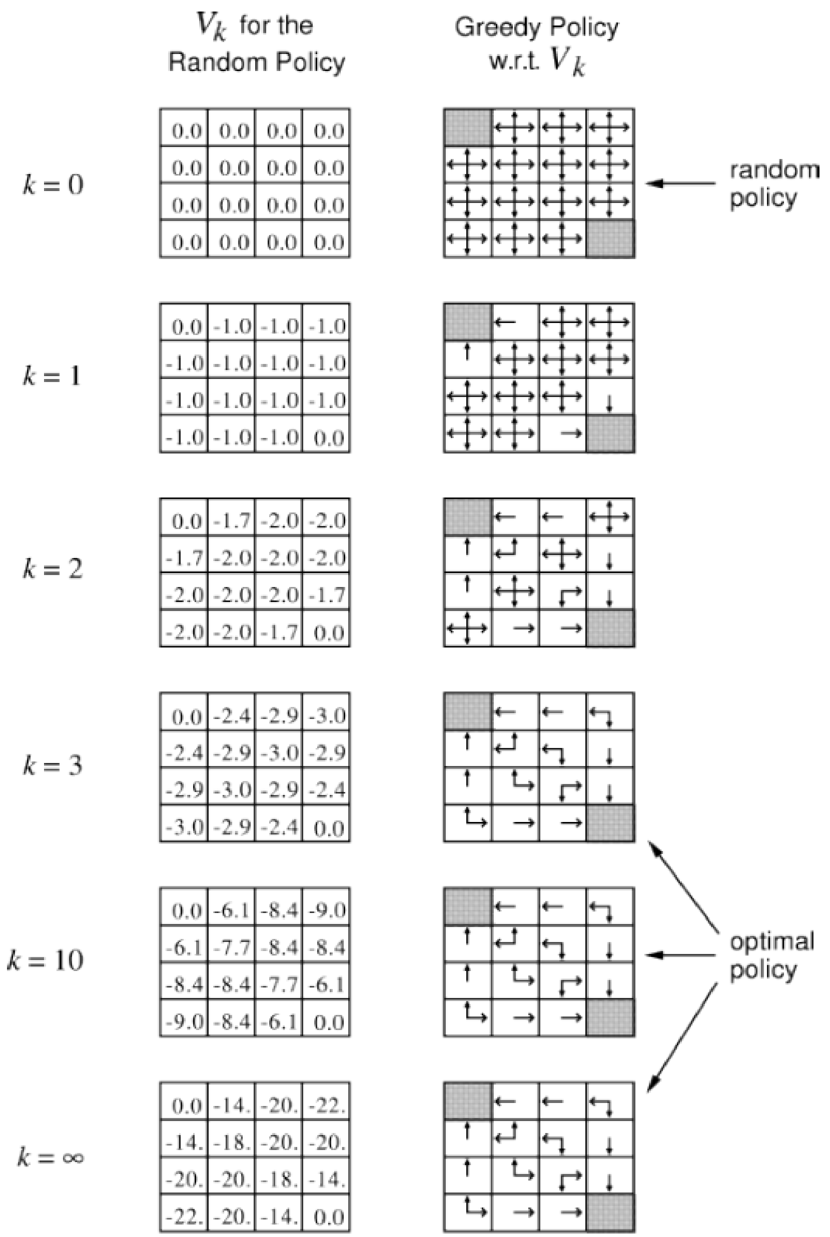


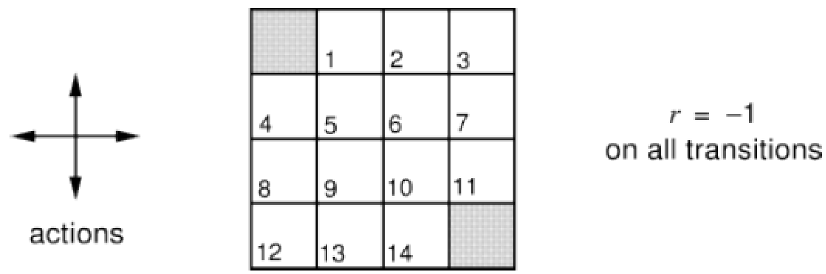
Sutton, Barto: Reinforcement Learning: An Introduction.

Policy iteration – PI

- Key idea: $\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \ldots \xrightarrow{I} \pi^* \xrightarrow{E} V^{\pi^*}$

1. policy π evaluation (E step):
   - find state values $V^\pi(s)$,
   - start: $V(s) = 0 \; \forall$ non-terminal states (V known in them),
   - iteration: until $V(s)$ gets steady ($\max_S |V_{k+1}(s) - V_k(s)| < \varepsilon$),
2. policy improvement $\pi \to \pi'$ (I step):
   - adapt to the new state values,
   - deterministic π: in every state takes single action,
     if $Q^\pi(s, \pi'(s)) \geq V^\pi(s)$ for $\forall s$, π' is not worse than π,
     obviously $\pi'(s) = \arg\max_a Q^\pi(s, a)$, chooses the currently best action,
   - stochastic π: action selection is driven by a probability distribution,
     the same logic except for $Q^\pi(s, \pi'(s)) = \sum_a \pi'(s, a) Q^\pi(s, a)$,
3. if π and π' differ in at least one state, go to step 1 with π'.





:: Random policy evaluation:

(greedy deterministic to illustrate only)

- $V(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right]$, $\pi(s,a) = 1/4, R^a_{ss'} = -1, P^a_{ss'} = 1, \gamma = 1$
- k=0: $\forall s V(s) = 0$, no change in terminal states
- k=1: $V(1) = V(2) = \ldots = V(14) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s') \right] = 4(1/4 * 1(-1 + 1 * 0)) = -1$
- k=2: $V(1) = 1/4(3 * 1 * (-1 + 1(-1)) + 1 * 1(-1 + 1 * 0)) = -7/4 = -1.75$
- k=3: $V(1) = 1/4(2 * 1 * (-1 + 1(-2)) + 1 * 1(-1 + 1(-1.75)) + 1 * 1(-1 + 1 * 0)) = -9.75/4 = -2.44$

Value iteration – VI

- is it necessary to evaluate/know the state values for the given policy perfectly?
  – late iterations often leave policy unchanged,
  – and may spend most of the time of the whole dynamic algorithm,
- **value iteration**
  – policy evaluation stopped after first iteration,
  – more frequent policy changes,
  – in some tasks faster convergence, but does not outperform PI in general,
- in terms of Bellman equation, a new iteration rule originates

$$V^{\pi}(s) = \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^{\pi}(s')\right]$$

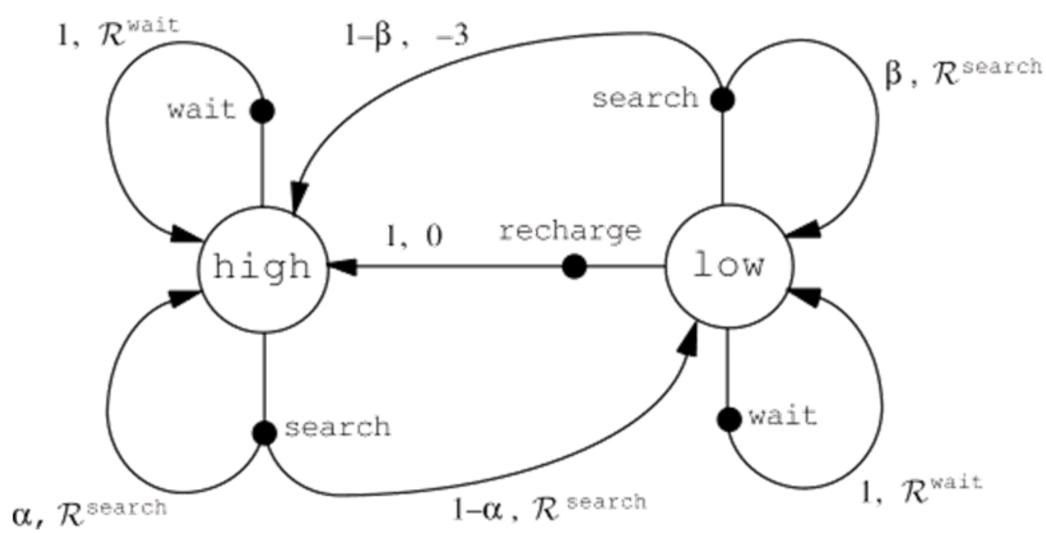
MDP – recycling robot

:: Mobile robot that cleans up/collects cans

- two internal states – battery low or high,
- three actions – search for cans, remain stationary, go to home base to recharge,
- positive reward for each can, negative reward when depleted needing rescue.

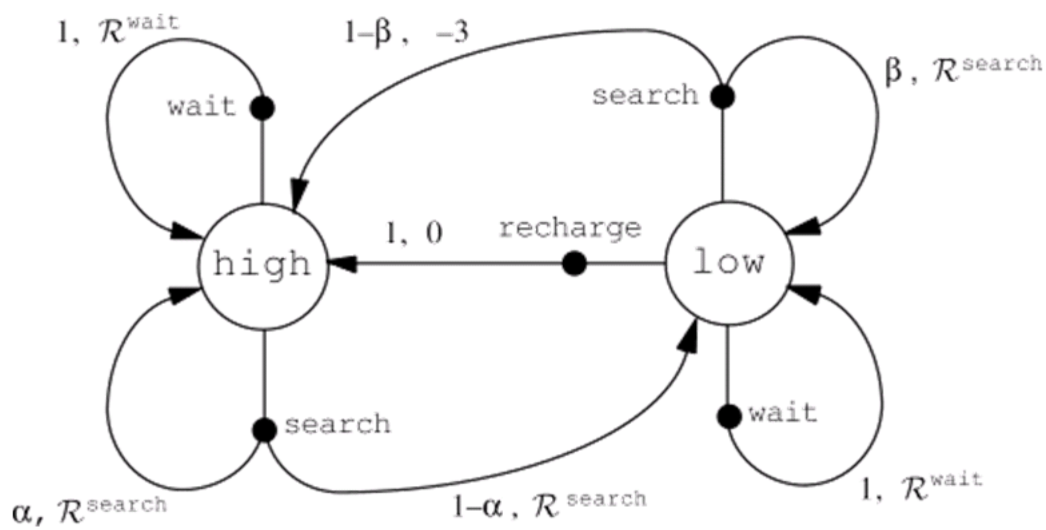
:: **Logical goal**: collect as many cans as possible without any external aid.

:: **Technical goal**: develop a policy that maximizes long term reward.



Sutton, Barto: Reinforcement Learning: An Introduction.

Recycling robot – DP solution



- Bellman equation: $V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s')\right]$
- Iteration equations for particular deterministic action (policy) choices high=h, wait=w, etc.:

$$\pi(h,w) = 1: \quad V(h) = Q(h,w) = R^w + \gamma V(h)$$
$$\pi(h,s) = 1: \quad V(h) = Q(h,s) = R^s + \gamma\left[\alpha V(h) + (1-\alpha)V(l)\right]$$
$$\pi(l,r) = 1: \quad V(l) = Q(l,r) = \gamma V(h)$$
$$\pi(l,w) = 1: \quad V(l) = Q(l,w) = R^w + \gamma V(l)$$
$$\pi(l,s) = 1: \quad V(l) = Q(l,s) = \beta R^s - 3(1-\beta) + \gamma\left[\beta V(l) + (1-\beta)V(h)\right]$$

Recycling robot – DP solution

:: **Parameters**: $\alpha = 0.95$, $\beta = 0.9$, $R^s = 2$, $R^w = 1$, $\gamma = 0.9$, $\varepsilon = 0.01$

:: **Method**: policy iteration (EI cycle)

1. Randomly choose a deterministic policy:
   $\pi(low, wait) = \pi(high, wait) = 1$,
2. set $V(low) = V(high) = 0$,
3. use the iteration equations until V values get steady,
4. use evaluations in V to determine optimal actions:
   $V(s) = \max_a Q^\pi(s, a)$, $\pi'(s) \approx \arg\max_a Q^\pi(s, a)$
5. in the case of no policy change stop, go to step 2 otherwise.

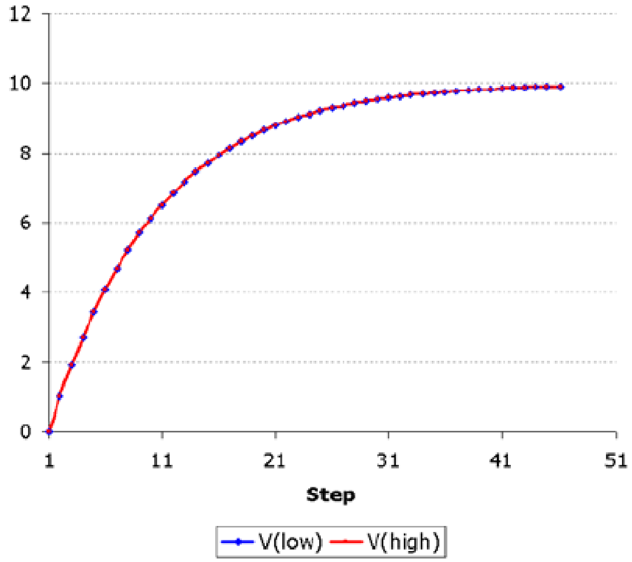
Initialize:

$\pi(l,w) = \pi(h,w) = 1$

$V(h) = R^w + \gamma V(h)$, $V(l) = R^w + \gamma V(l)$

Evaluation 1:

$V(h) = V(l) = 10$, 46 steps



Improvement 1:

$\pi(l,s) = \pi(h,s) = 1$

$V(h) = R^s + \gamma[\alpha V(h) + (1-\alpha)V(l)]$

$V(l) = \beta R^s - 3(1-\beta) + \gamma[\beta V(l) + (1-\beta)V(h)]$

Evaluation 2:

$V(h) = 19$, $V(l) = 16.8$, 52 steps

Improvement 2:

$\pi(l,r) = \pi(h,s) = 1$

$V(h) = R^s + \gamma[\alpha V(h) + (1-\alpha)V(l)]$

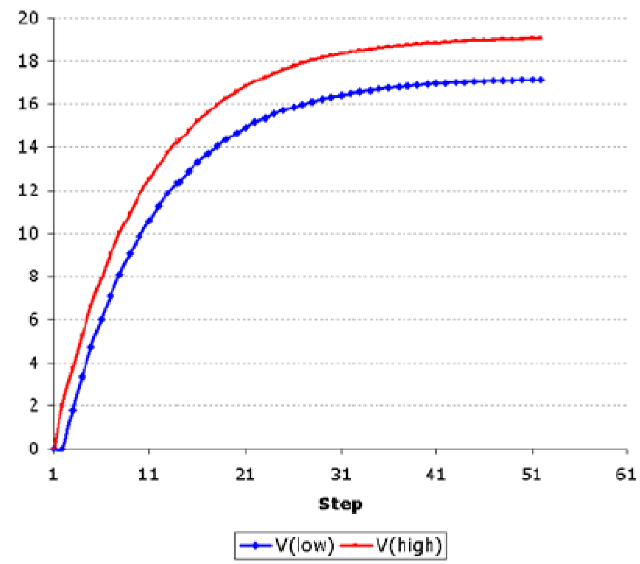
$V(l) = \gamma V(h)$

Evaluation 3:

$V(h) = 19.1$, $V(l) = 17.1$, 52 steps

Improvement 3:

$\pi(l,r) = \pi(h,s) = 1 \rightarrow$ STOP



SUMMARY:

policy: low $\rightarrow$ recharge, high $\rightarrow$ search

$V(h) = 19.1$, $V(l) = 17.1$, 150 iterations

Recycling robot – DP solution

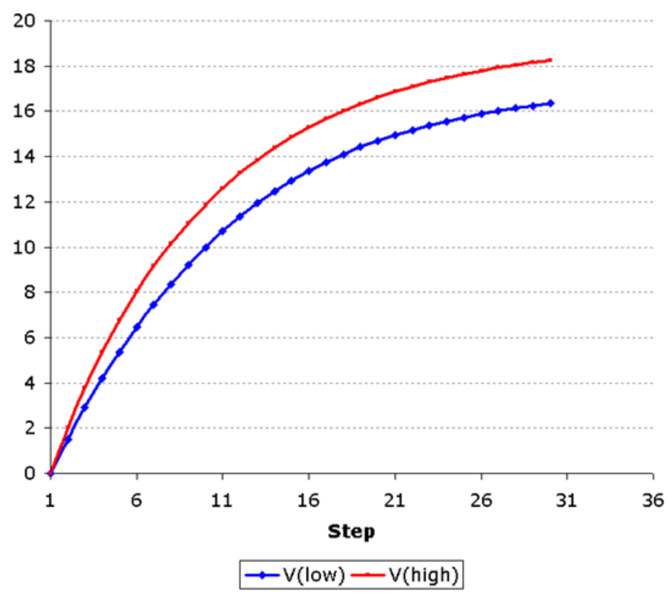
:: **Parameters**: $\alpha = 0.95$, $\beta = 0.9$, $R^s = 2$, $R^w = 1$, $\gamma = 0.9$, $\varepsilon = 0.01$,

:: **Method**: value iteration

1. Set $V(low) = V(high) = 0$.
2. Use evaluations in V to determine optimal actions:
   $V(s) = max_a Q^\pi(s,a)$, $\pi'(s) \approx \arg\max_a Q^\pi(s,a)$.
3. apply once the current best actions and recompute values in $V(s)$,
4. in the case of no state value change larger than ε stop,
   go to step 2 otherwise.

Recycling robot – DP solution

step 0: $V(l) = V(h) = 0$, $\pi(l, s) = \pi(h, s) = 1$
step 1: $V(l) = 1.5$, $V(h) = 2$, $\pi(l, s) = \pi(h, s) = 1$
step 9: $V(l) = 9.2$, $V(h) = 11.1$, $\pi(l, r) = \pi(h, s) = 1$,
policy change
step 52: $V(l) = 17.1$, $V(h) = 19.1$, $\pi(l, r) = \pi(h, s) = 1$,
all V perturbations smaller than ε, STOP



SUMMARY:
policy: low $\rightarrow$ recharge, high $\rightarrow$ search
$V(h) = 19.1$, $V(l) = 17.1$, 52 iterations

Why does value iteration certainly converge?

- contraction $c(x)$
  - $\exists k \, \forall x_1 x_2 : d(c(x_1) - c(x_2)) \leq k d(x_1 - x_2)$,
  - d is a metric (distance function), constant $0 \leq k < 1$,
  - fixed point b_c: $c(b_c) = b_c$, $c(c(\ldots c(x))) = b_c$,
  - each contraction has only one fixed point,
  - example: $c(x) = \frac{x}{2}$, $d(x, y) = |x - y|$, $b_c = 0$,
- value iteration equation
  - $V_{i+1}(s) = \max_a \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V_i(s')\right]$
  - can be simplified as $V_{i+1} \leftarrow BV_i$,
  - as d we employ max norm $||V|| = max_s |V(s)|$,
- the above defined B is wrt $||\ ||$ contraction (without proof)
  - $||BV_i - BV_i'|| \leq \gamma ||V_i - V_i'||$.

Why does value iteration certainly converge?

- provided that B is a contraction wrt $|| \, ||$
  - for any pair of state utility vectors it holds

    $$||BV_i - BV_i'|| \leq \gamma ||V_i - V_i'|| \Rightarrow ||V_{i+1} - V_i|| \leq \gamma ||V_i - V_{i-1}||,$$

    * value iteration converges for $\gamma < 1$,
  - the fixed point is the actual state utility vector V^*
    * $||BV_i - V^*|| \leq \gamma ||V_i - V^*||$,
    * converges exponentially with γ.

Summary

- MDPs allow to search stochastic state spaces
  - computational complexity is increased due to stochasticity,
- problem solving = policy finding
  - policy assigns each state the optimal action, can be stochastic too,
  - basic approaches are policy iteration and value iteration,
  - other choices can be modified iteration approaches, possibly asynchronous,
- techniques similar to MDP
  - POMDP for partially observable environments,
  - RL for environments with unknown models,
- applications
  - agent technology in general, robot control and path planning in robotics,
  - network optimization in telecommunication, game playing.

Recommended reading, lecture resources

:: Reading

- Russell, Norvig: **AI: A Modern Approach**, Making Complex Decisions
  - chapter 17,
  - online on Google books: http://books.google.com/books?id=8jZBksh-bUMC,
- Sutton, Barto: **Reinforcement Learning: An Introduction**
  - MIT Press, Cambridge, 1998,
  - http://www.cs.ualberta.ca/~sutton/book/the-book.html.

Demo

- RL simulator
  - find the optimal path in a maze
  - implemented in Java
  - http://www.cs.cmu.edu/~awm/rlsim/