RL 8: Value Iteration and Policy Iteration

Michael Herrmann

University of Edinburgh, School of Informatics

05/02/2016

Last time: Eligibility traces: TD(λ)

Determine the δ error:

$$\delta_{t+1} = r_t + \gamma \hat{V}_t(s_{t+1}) - \hat{V}_t(s_t)$$

Update all states the agent has visited recently:

$$\hat{V}_{t+1}(s) = \hat{V}_t(s) + \eta \delta_{t+1} e_{t+1}(s)$$

Update the eligibility traces to make sure that the update becomes weaker and weaker since the state has been visited the last time:

$$e_{t+1}(s) = \begin{cases} 1+\gamma\lambda e_t(s) & \text{if } s = s_t \\ \gamma\lambda e_t(s) & \text{if } s \neq s_t \end{cases}$$

Parameters:

- $0 < \gamma \lesssim 1$ discount factor $(1-\gamma)^{-1}$ time horizon
- $\eta \gtrsim 0$ learning rate (Robbins-Monro condition)
- $\lambda \lesssim 1$ eligibility trace parameter
- ε (or β) exploration rate (or temperature)

Monte-Carlo methods

MC

- Solving reinforcement learning problems based on averaging sample returns
- For episodic problems
- No model required

RL

- Shares embeddedness
- Weights by recency

Monte-Carlo methods (First visit MC)

- Initialise
  - $\pi :=$ policy to be evaluated
  - $V :=$ an arbitrary state-value function
  - Returns(s):= an empty list for all $s \in \mathcal{S}$
- Repeat
  - Generate an episode using π
  - For each state s appearing in the episode
    - $G :=$ return following the first occurrence of s
    - Append G to Returns(s)
    - $V(s) :=$ average(Returns(s))

Monte-Carlo control algorithm

- Initialise for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  - $\pi :=$ arbitrary policy
  - $\mathcal{Q}(s,a) :=$ an arbitrary state-action value function
  - Returns(s,a):= an empty list
- Repeat
  - Choose s_0 and a_0
  - Generate an episode starting from s_0 and a_0 using π
  - For each pair s and a appearing in the episode
    - $G :=$ return following the first occurrence of s and a
    - Append G to Returns(s,a)
    - $\mathcal{Q}(s,a) :=$ average(Returns(s,a))
  - For each s in the episode
    $\pi(s) := \arg\max_a \mathcal{Q}(s,a)$

Discussion: MC

- MC averages return over episodes
- Episodes must terminate for every policy and must reach all state-action pairs
- Sufficient exploration crucial
- MC methods (usually) do not bootstrap (values are not updated based on other values)
- Available as on-policy or off-policy variants

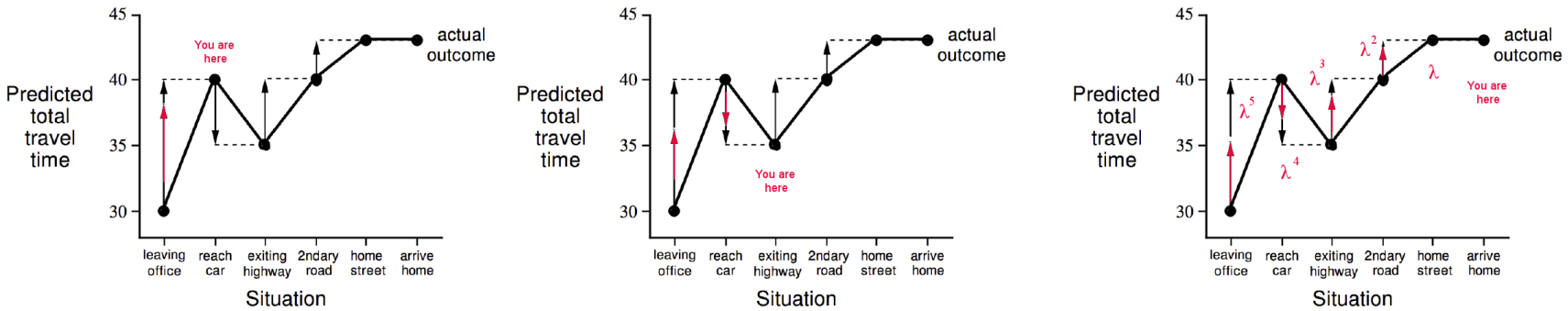
Relation to TD with eligibility traces

- TD(λ) provides efficient estimates of the value function in Markovian reward processes
- It generalises Monte Carlo methods, but can be used as well in non-episodic tasks
- Appropriate tuning of λ (and γ) can lead to a significantly faster convergence than both Monte Carlo and TD(0).

Example: Driving home

State	Elapsed Time (minutes)		Predicted Time to Go	Predicted Total Time
leaving office	0		30	30
reach car, raining	5	(5)	35	40
exit highway	20	(15)	15	35
behind truck	30	(10)	10	40
home street	40	(10)	3	43
arrive home	43	(3)	0	43

Eligibility Traces



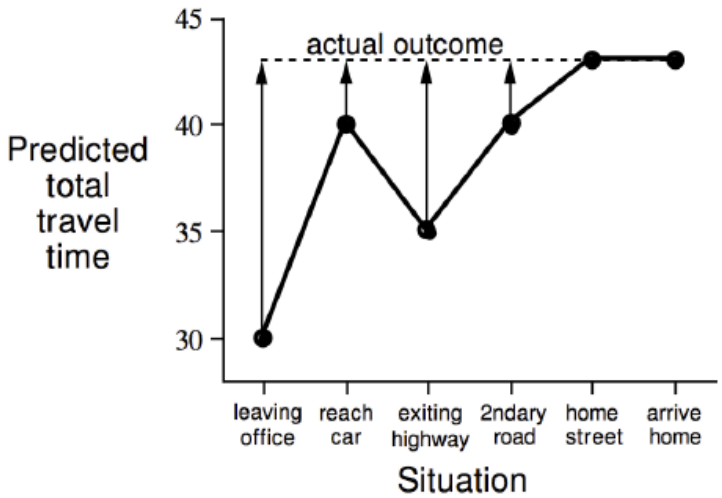
...

Changes when using TD with eligibility traces.

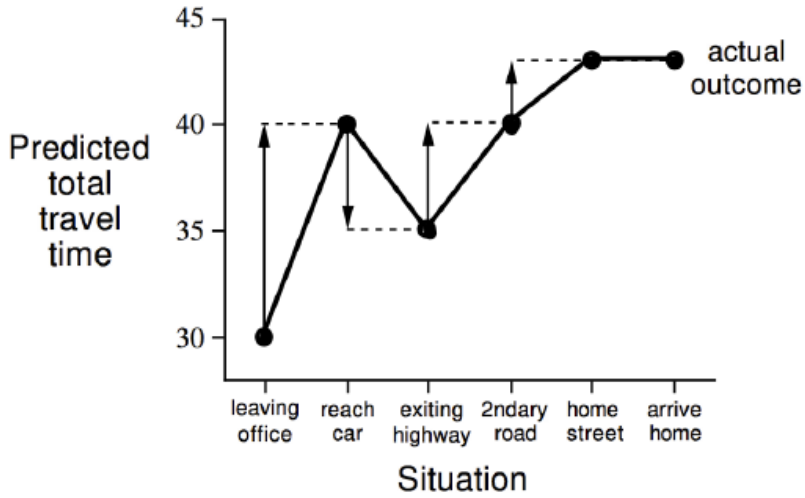
(This illustration is not to scale and does not reflect the accumulation of the changes)

Comparison: MC vs. TD

Expected time (= cost = neg. reward) to arrive home:

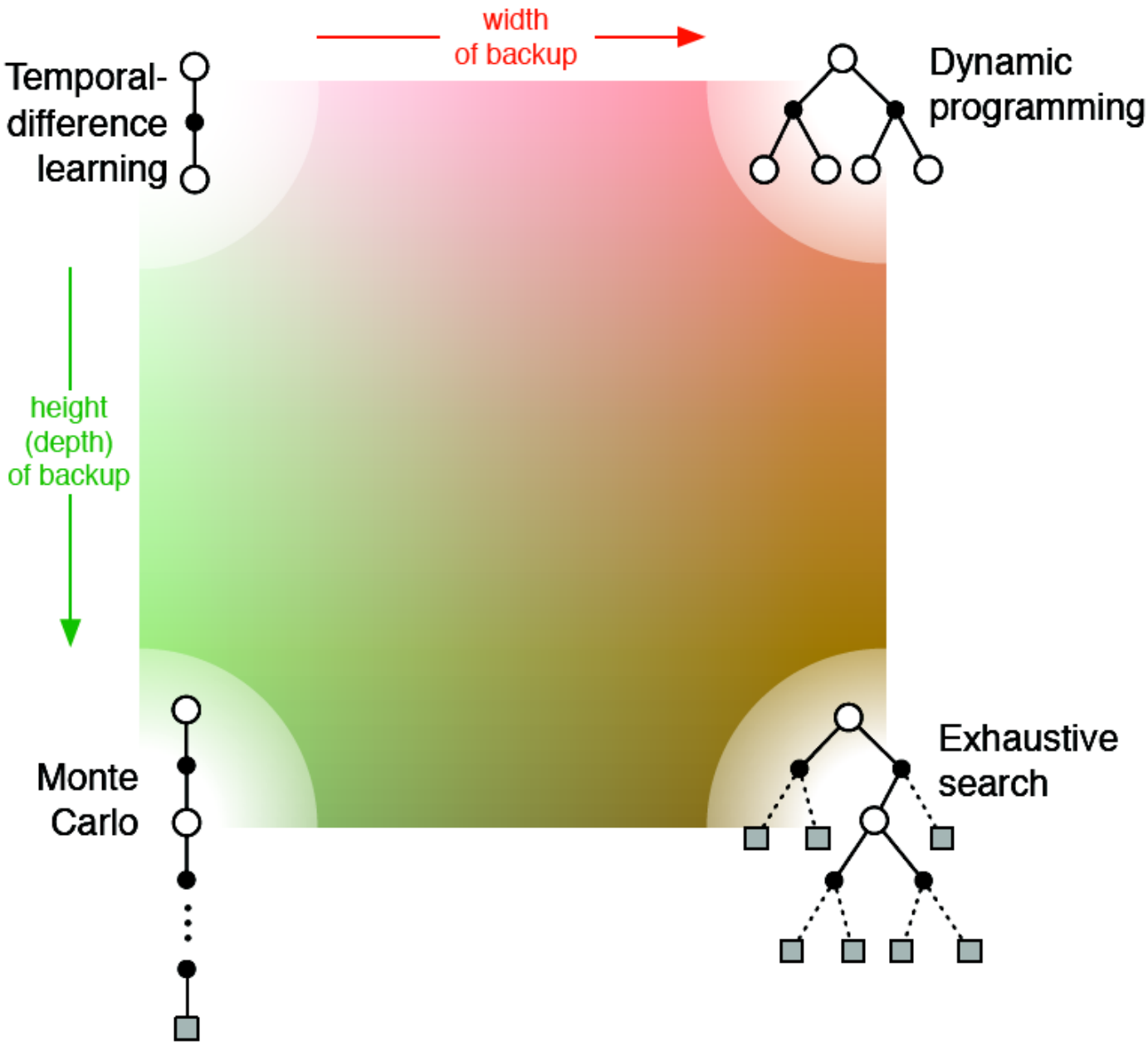


Changes implied by MC
(once at the end of episode)



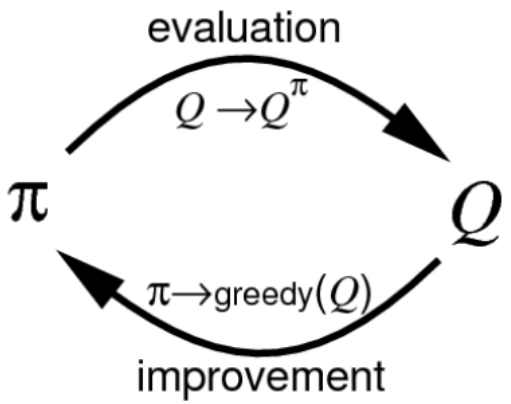
Changes implied by TD(0)
(updating one value per step)

DP, MC and RL


width
of backup
Temporal-
difference
learning
Dynamic
programming
height
(depth)
of backup
Monte
Carlo
Exhaustive
search

Overview: MC, DP and TD(λ)

- Dynamic programming
  - Value iteration
  - Policy iteration
  - Mixed schemes
- Back to TD(λ)
- Outlook to RW applications of RL



Dynamic Programming

DP

- The art of solving problems by breaking them into subproblems
- Still computationally expensive
- Perfect model required ("certainty equivalence assumption")

RL

- Uses a similar approach but does not require a model and cannot rely on many data,
- Value functions used in the the way
- DP is of fundamental theoretical importance to RL

Dynamic Programming: Value iteration

- initialise[1] an array $V(s) := 0$, for all $s \in \mathcal{S}$
- set error threshold θ to a small positive number[2]
- repeat[3]
  - $\Delta := 0$
  - For each $s \in \mathcal{S}$
    - temp $:= V(s)$
    - $V(s) := \max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V(s')\right]$
    - $\Delta := \max\{\Delta, |\text{temp} - V(s)|\}$
- until $\Delta < \theta$
- $\pi(s) := \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V(s')\right]$

---

[1] Other initialisation schemes may be preferable, e.g. for tie breaking.

[2] θ is a criterion for convergence, repeated tests may be useful here.

[3] In RL, the average over s' is represented by actual sampling at small ε.

Remark (fixed policy π)

- Value determination possible by iteration, but sometimes also by direct computation (given the transition probabilities M^π):

$$V(i) = r(i) + \gamma \sum_j M_{ij}^{\pi(i)} V(j) \quad \forall i \in S$$

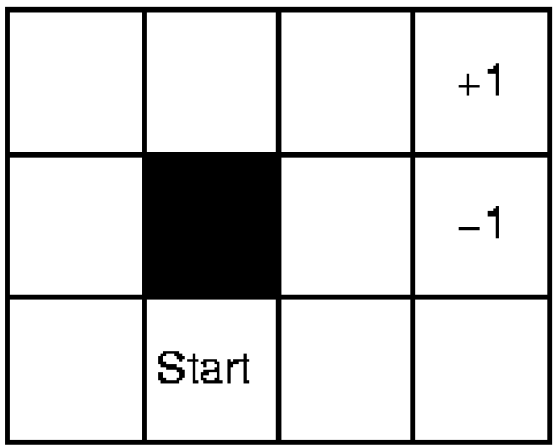
  is a system of linear equations with dimension $|S|$:

$$V = (I - \gamma M^\pi)^{-1} r$$

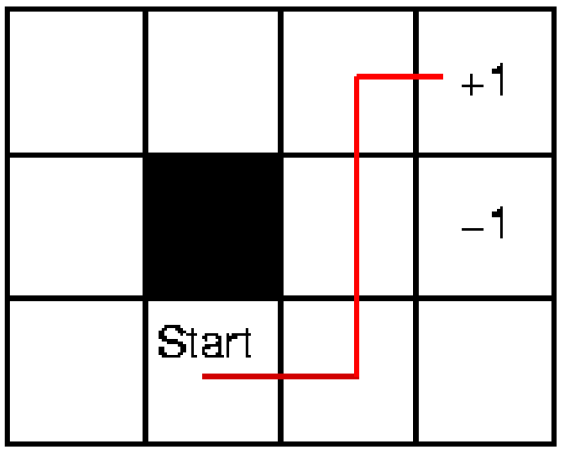
  The system does not necessarily have a solution. Theory, therefore, often assumes $\gamma < 1$.
- For real-time applications even MDPs are hard to compute: Try deterministic approximations.

Example from Stachnis/Burgard

A robot in a small grid world with "reflecting walls"



Reward:
$r = \pm 1$ as shown or
$r = -0.04$ per step



Optimal path from
Start to Goal
avoiding the trap
(deterministic case)

The optimal policy in a non-deterministic problem

Optimal policy

$$\pi^*(i) = \arg\max_a \sum_j M_{ij}^a V(j)$$

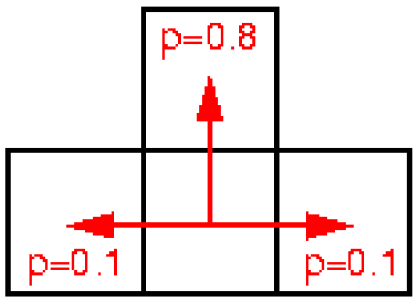
M_{ij}^a: Probability of reaching state j form state i with action a.

$V(j)$: Value of state j.

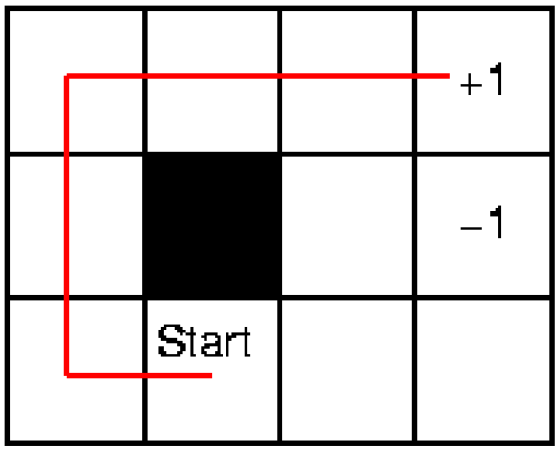
Given the value function and the transition probabilities (Markovian!), we can easily calculate the optimal policy

We know already a way to obtain the value function.

Same example in a non-deterministic world

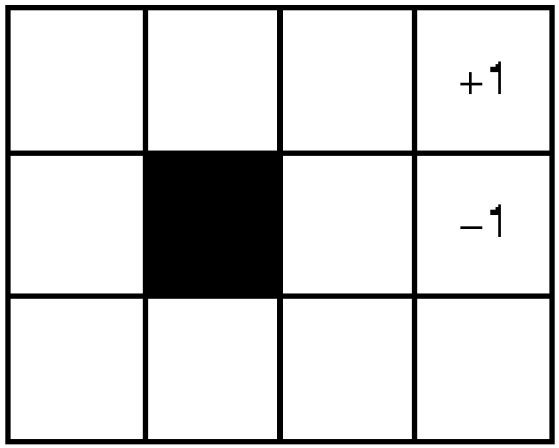


Robot does not exactly perform the desired action

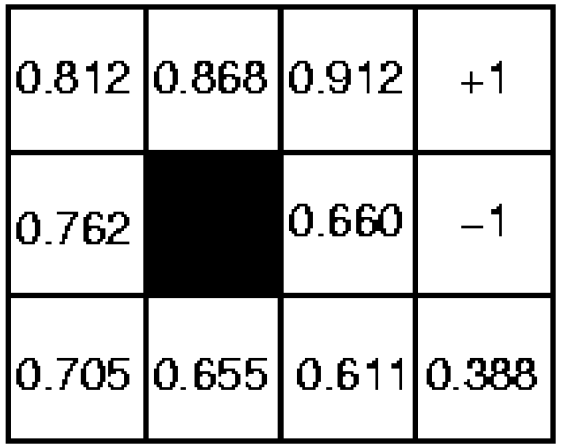


If cost per step is low it pays to choose a safer path

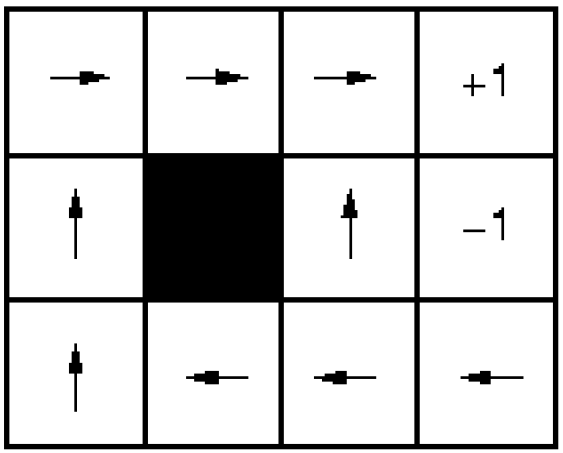
Value Iteration and Optimal Policy



1. Given environment

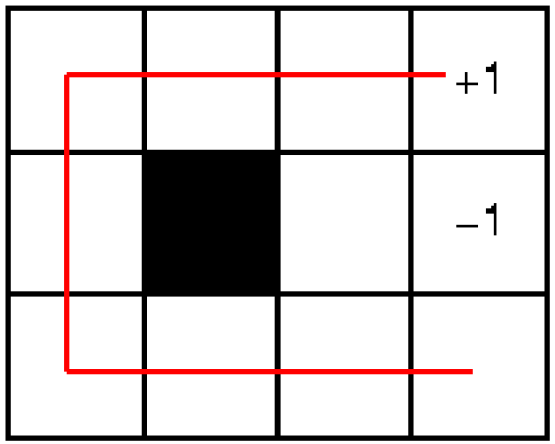


2. Calculate state values



3. Extract optimal policy

(now for an arbitrary starting position)



4. Execute actions

Optimality

Note that state (3,2) has higher value than (2,3), but policy of (3,3) points to (2,3).

Policy is not the gradient of the value function!

$$\pi^*(i) = \arg\max_a \sum_j M_{ij}^a V(j)$$

Obviously, policy may converge faster than the values of the value function.

For the policy to converge it is sufficient that the ***relations*** between the values are correct.

Can the we compute the optimal policy in a faster way, i.e. without calculating optimal values first?

Dynamic Programming: Policy Evaluation

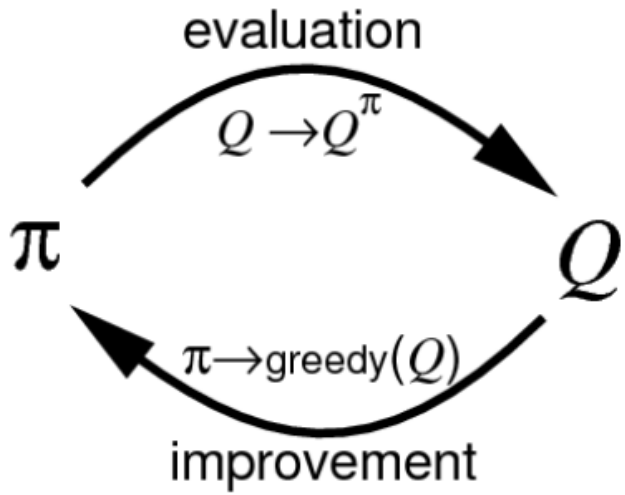
- input π, the policy to be evaluated
- initialise an array $V(s) := 0$, for all $s \in \mathcal{S}$
- set error threshold θ to a small positive number
- repeat
  - $\Delta := 0$
  - For each $s \in \mathcal{S}$
    - temp $:= v(s)$
    - $V(s) := \pi(a|s) \sum_{s'} p(s'|s,a) \left[r(s,a,s') + \gamma V(s') \right]$
    - $\Delta := \max \{\Delta, |\text{temp} - V(s)|\}$
- until $\Delta < \theta$
- output $V \approx V^{\pi}$

Dynamic Programming: Policy improvement

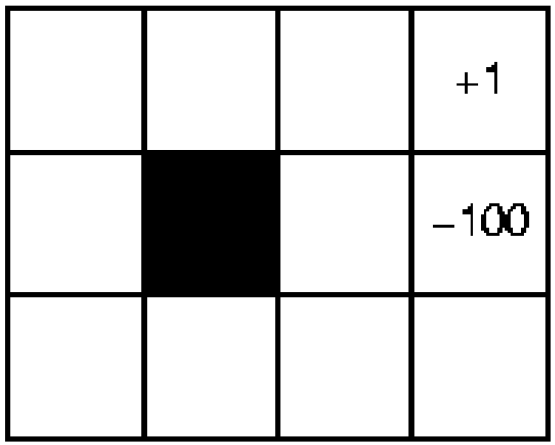
- policy-stable := true
- for each $s \in \mathcal{S}$
  - temp := $\pi(s)$
  - $\pi(s) := \arg\max_a \sum_{s'} p(s'|s,a)\left[r(s,a,s') + \gamma V(s')\right]$
  - if temp $\neq \pi(s)$, then policy-stable := false
- return π, policy-stable

Dynamic Programming: Policy iteration

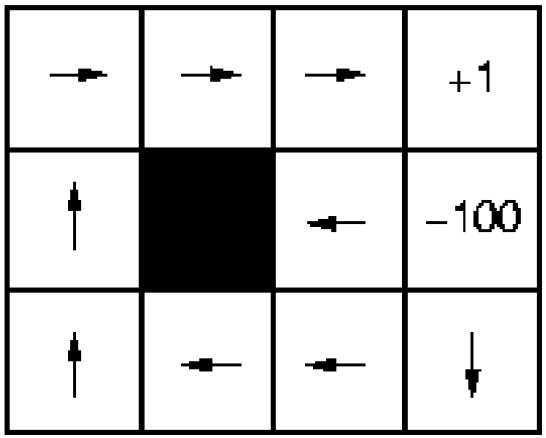
- Initialisation $V(s)$ and $\pi(s)$ for all $s \in \mathcal{S}$
- Repeat
  - Policy evaluation (until convergence)
  - Policy improvement (one step)
- until policy-stable
- return π and V (or $\mathcal{Q}$)



Value iteration vs. Policy Iteration: Example (variation)



Problem



Solution (policy iteration)

Given the (0.1, 0.8, 0.1)-probability model it is optimal to try to move from (4,3) and (3,2) by bumping to the walls.

Then, entering the trap at (4,2) has probability 0.

Value iteration gave "up" for (3,2) and "left" for (4,3).

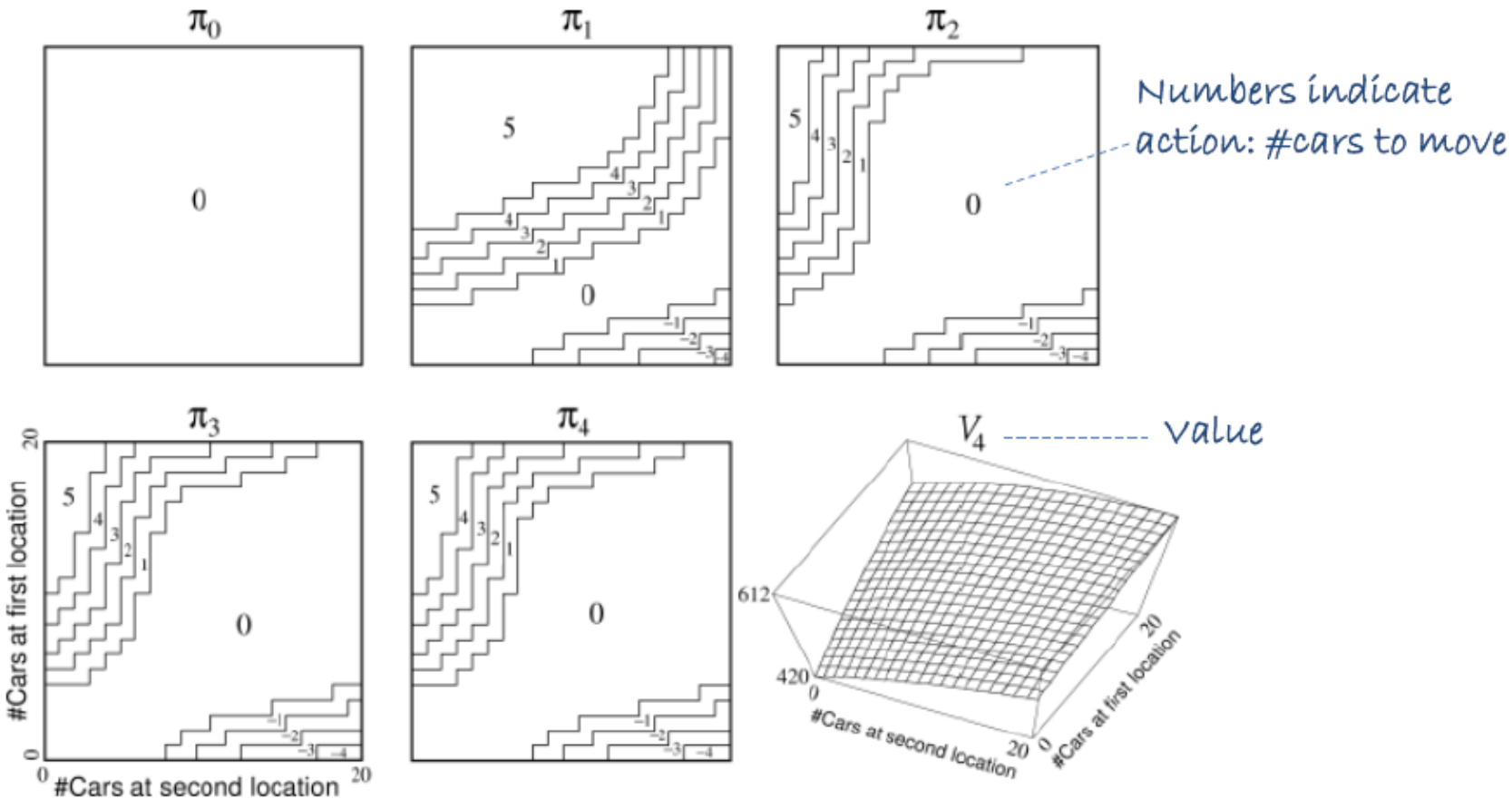
Example: Jack's car rental (4.2 in S+B)

- $10 for each car rented (must be available when request received)
- Two locations, maximum of 20 cars at each
- Cars returned and requested randomly
  - Poisson distribution, n returns/requests with probability $\frac{\lambda^n}{n!}e^{-\lambda}$
  - Location 1: Average requests = Average returns = 3
  - Location 2: Average requests = 4, Average Returns = 2
- Can move up to 5 cars between locations overnight (costs $2 each)

Problem setup:

- States, actions, rewards?
- Transition probabilities?

Jack's car rental: Solution by policy iteration



Jack's car rental: Discussion

Difficult to adapt the solution to different conditions, e.g.

- Suppose first car moved is free but all other transfer cost \$2
  - From location 1 to location 2 (not other direction!)
  - Because an employee would anyway go in that direction, by bus
- Suppose only 10 cars can be parked for free at each location
  - More than 10 incur fixed cost of \$4 for using an extra parking lot

For more information see: Sutton & Barto

Discussion

- Value iterations turns Bellman optimality equation into an update rule
- Policy iteration: Try to get a better policy, using the currently best evaluation
  - Policy evaluation (until convergence)
  - Policy improvement (one step)
- Value iteration: Try to get a better evaluation, but use the best available policy
  - Policy evaluation (one step)
  - Policy improvement (one step)
- Combinations are possible:
  - truncated policy iteration
  - combinations of max and mean during evaluation step
- Generalised Policy Iteration

Conclusions on PI vs. VI

"Therefore in practice, value iteration should never be used. Implementation of modified policy iteration requires little additional programming effort yet attains superior convergence". Puterman, 1994

The policy iteration is polynomial in the problem size, but Leslie Pack Kaelbling remarks that: "In the worst case the number of iterations grows polynomially in $(1-\gamma)^{-1}$, so the convergence rate slows considerably as the discount factor approaches 1."

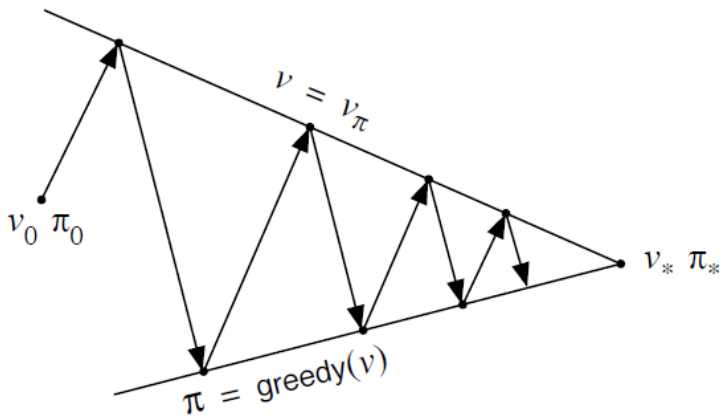
(M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving Markov decision problems. Proc. 11th Ann. Conf. on Uncertainty in AI, 1995.)

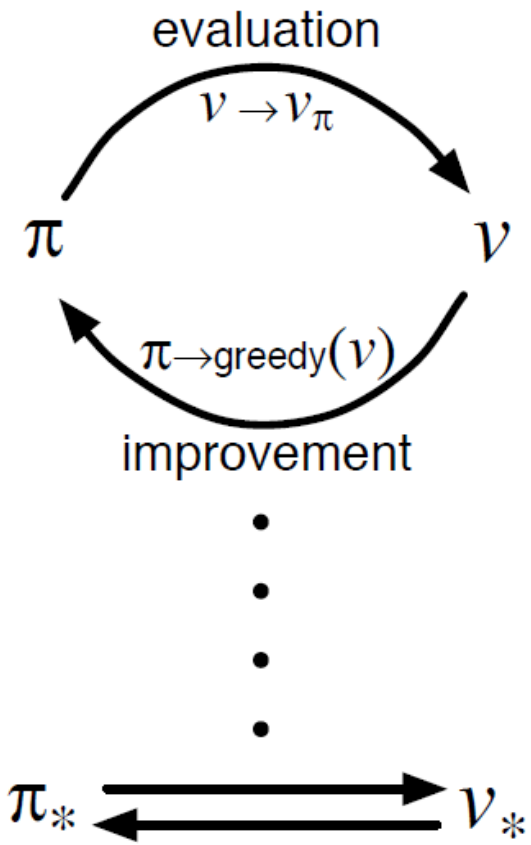
Philosophical arguments: E. B. Baum asks in *What is Thought?* (MIT, 2004) "What it fundamentally wrong with value iteration?"

Yet, value iteration is a straight-forward generalisation of the deterministic case. It may be more robust in dynamic problems, for higher uncertainty, or strong randomness.

Generalised Policy Iteration

- policy evaluation and policy improvement processes interact
- either complete convergence or truncated





- Similar to EM algorithms

Combined Value-Policy Iteration

E. Pashenkova (1996)

1. Perform Value Iteration and compute policy at each step of VI
2. IF no change in policy on two successive steps, fix the policy and perform one step of Policy Iteration:
    1. Value Determination finding precise values for the fixed policy;
    2. policy evaluation
    3. IF no change in policy, return it as an optimal policy, ELSE go to 1.

Outlook: Towards practical problems

- efficient state representations
  - hierarchical
  - options
  - function approximations
- efficient policy representations
- efficient algorithms

Many slides are adapted from web resources associated with Sutton and Barto's Reinforcement Learning book

. . . before being used by Dr. Subramanian Ramamoorthy in this course in the last three years.

Please check again the more formal considerations in the book *Algorithms for Reinforcement Learning* by C. Szepesvari, Chapters 2.1 and 4.1.

The first example today was adapted form " Autonomous Mobile Systems: The Markov Decision Problem Value Iteration and Policy Iteration by Cyrill Stachniss and Wolfram Burgard. These authors acknowledge: Russell & Norvig: AI – A Modern Approach (Chapter 17, pages 498ff)-