ECE 5984
Dynamic Programming

Jason J. Xuan, Ph.D.

Department of Electrical & Computer Engineering
Virginia Tech

Outline

Dynamic Programming?

Dynamic sequential or temporal component to the problem

Programming optimising a "program", i.e. a policy

- c.f. linear programming

- A method for solving complex problems
- By breaking them down into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems

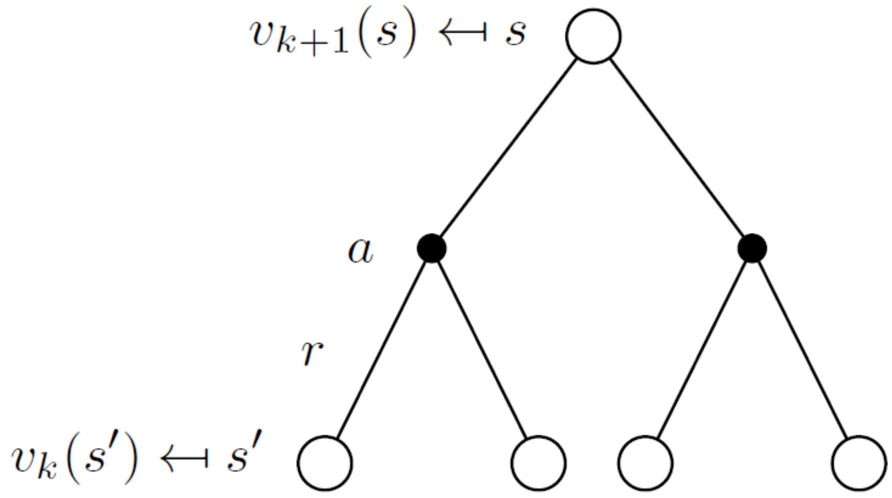
Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP
- For prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
  - or: MRP $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$
  - Output: value function v_{π}
- Or for control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function v_{*}
  - and: optimal policy π_{*}

Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_\pi$
- Using *synchronous* backups,
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
  - where s' is a successor state of s

Iterative Policy Evaluation (cont'd)

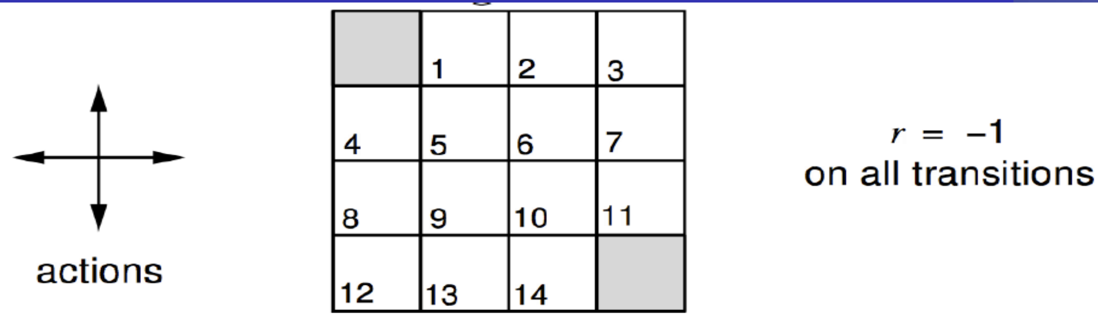


$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{\pi}} \mathbf{v}^k$$

Example: Small Grid

Evaluating a Random Policy in the Small Gridworld



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states 1, ..., 14
- One terminal state (shown twice as shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

Example: Iterative Value Evaluation

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

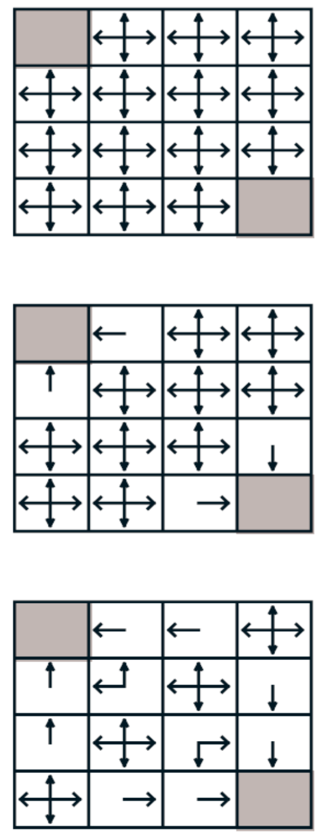
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



random policy

Example: Iterative Value Evaluation (cont'd)

$k = 3$

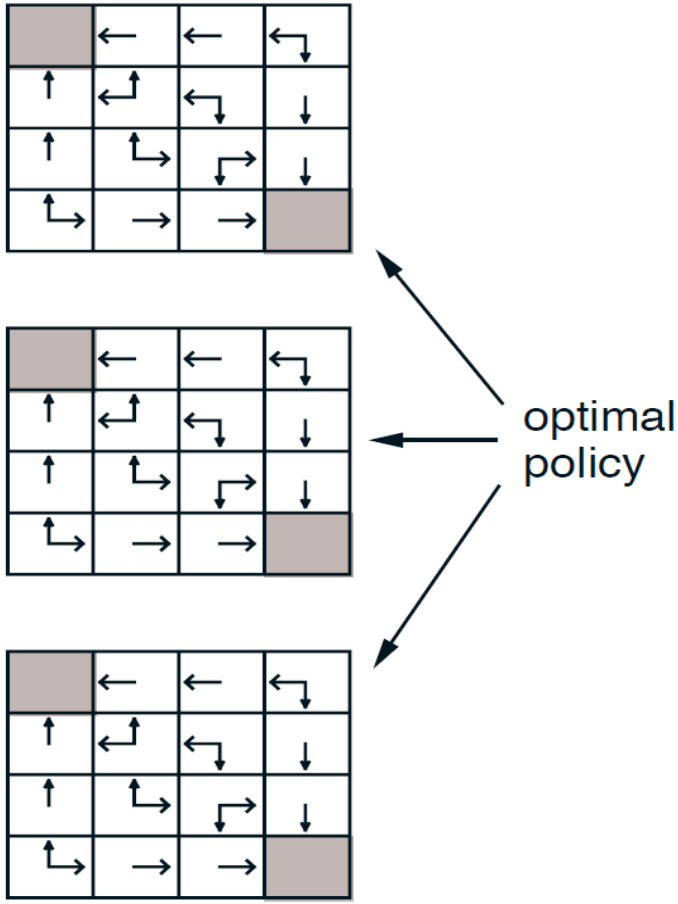
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



How to Improve a Policy

- Given a policy π
  - Evaluate the policy π

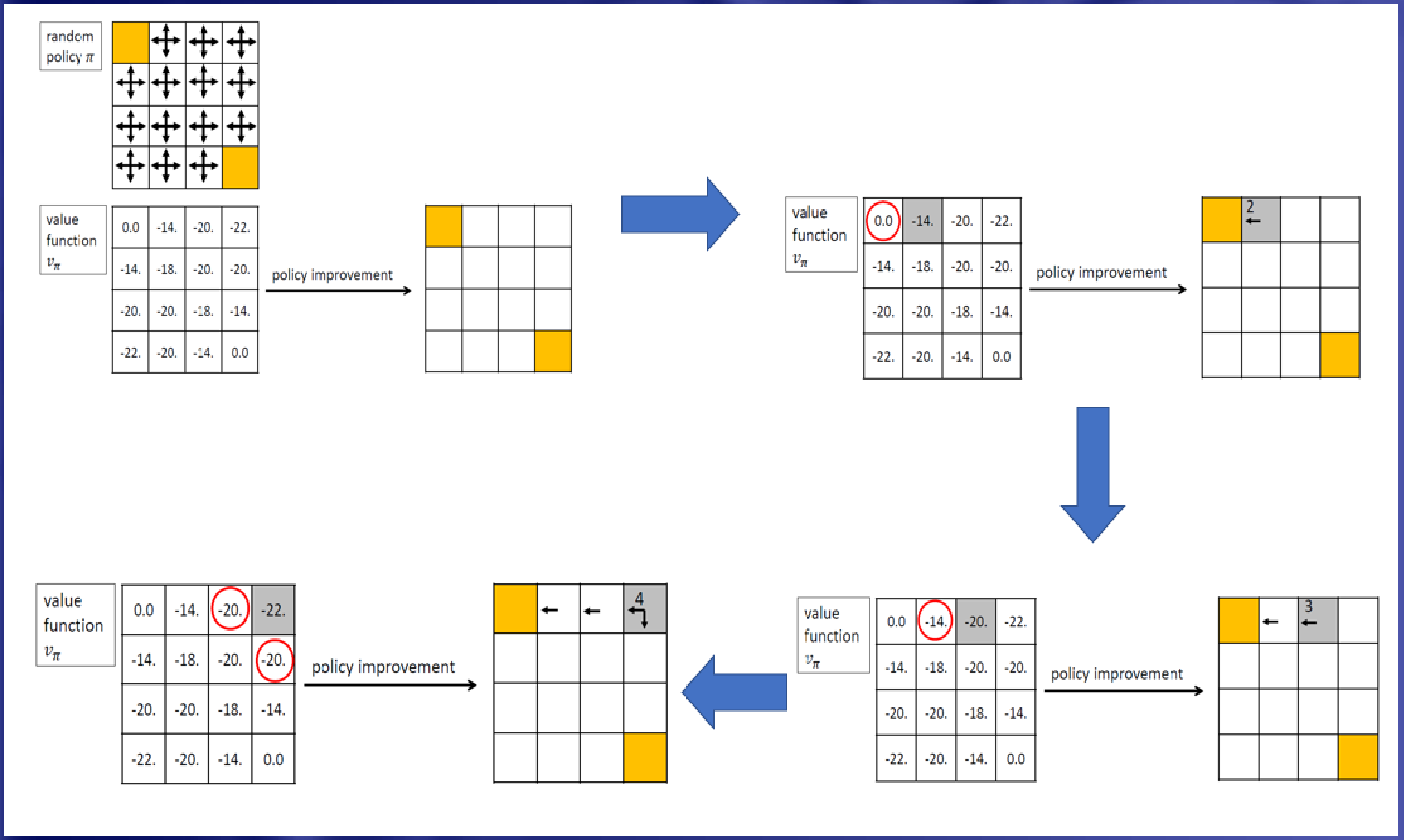
$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ... | S_t = s\right]$$

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to $\pi*$

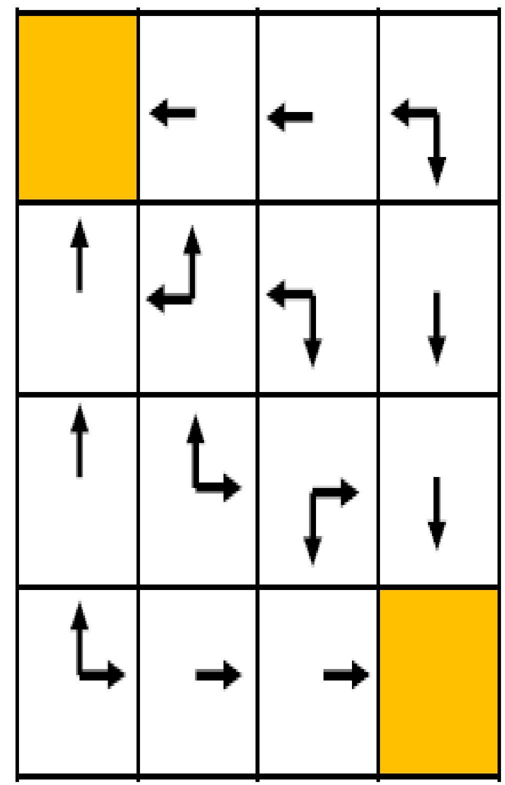
Policy Improvement



Policy Improvement (cont'd)

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

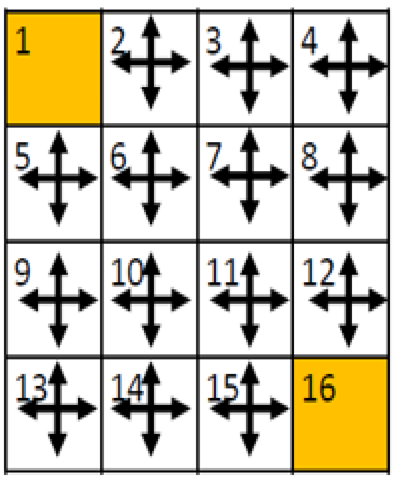
policy improvement



Policy Iteration

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

π_0



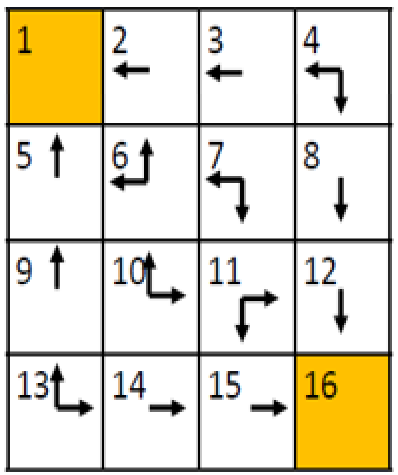
$\xrightarrow{E}$

v_{π_0}

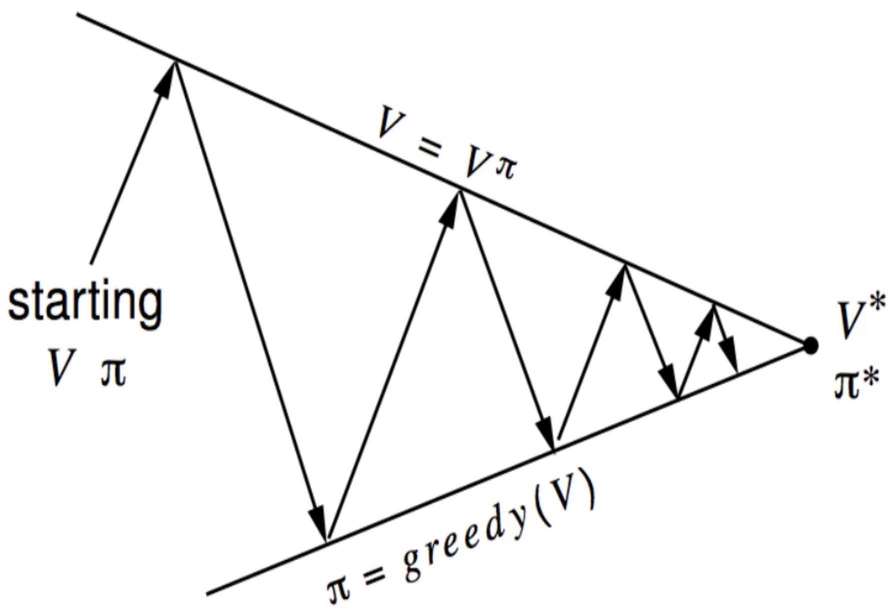
0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

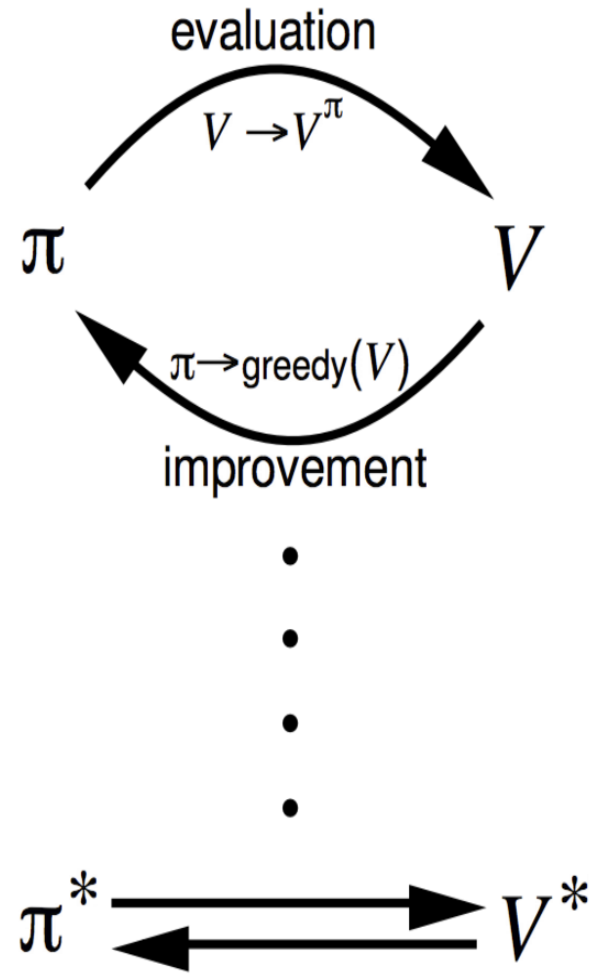
$\xrightarrow{I}$

π_1



Policy Iteration





Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

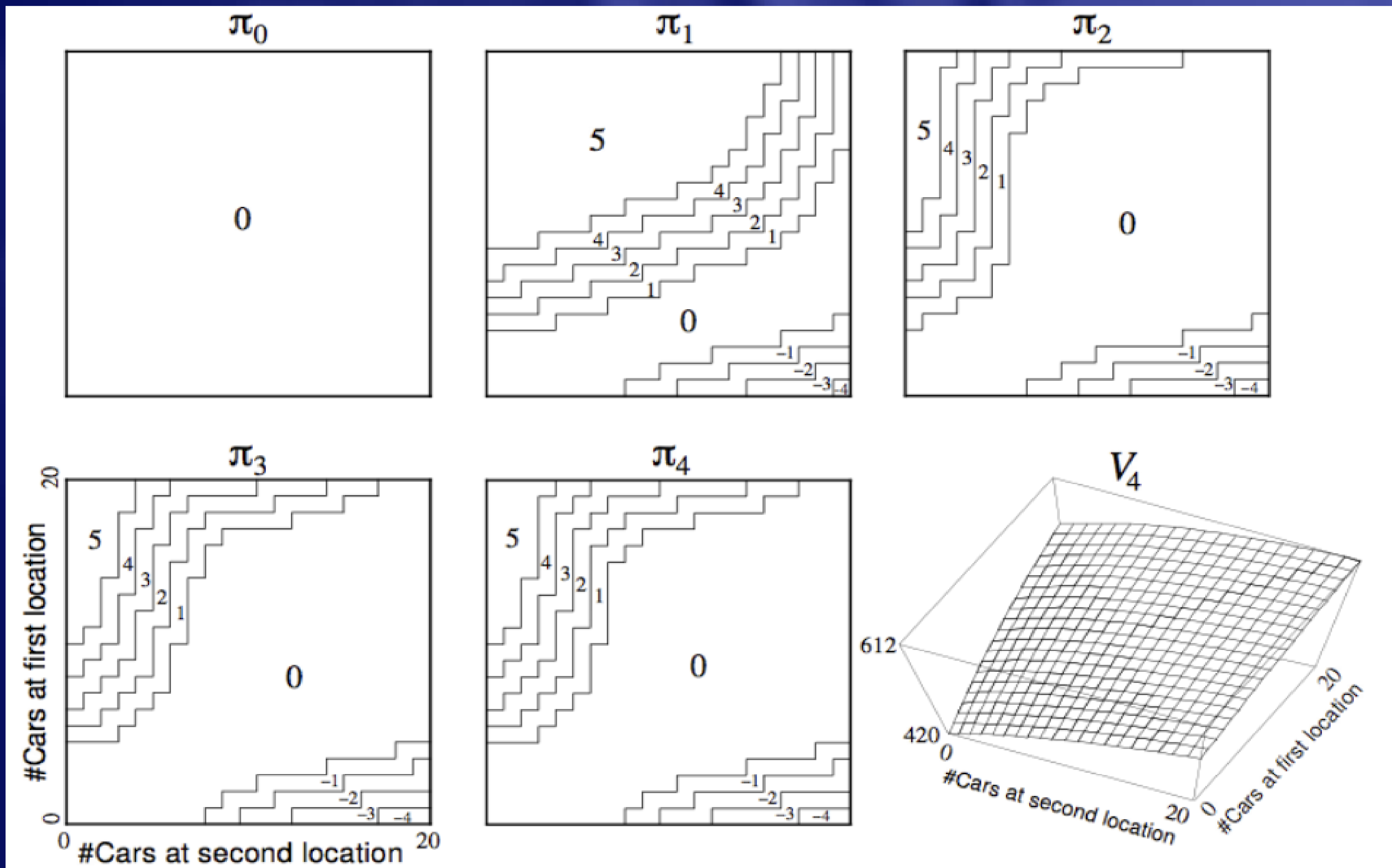
Greedy policy improvement

Example: Car Rental

- States: Two locations, maximum of 20 cars at each
- Actions: Move up to 5 cars between locations overnight
- Reward: \$10 for each car rented (must be available)
- Transitions: Cars returned and requested randomly
  - Poisson distribution, n returns/requests with prob $\frac{\lambda^n}{n!}e^{-\lambda}$
  - 1st location: average requests $= 3$, average returns $= 3$
  - 2nd location: average requests $= 4$, average returns $= 2$

Example: Car Rental

Policy Iteration (γ=0.9)



Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can *improve* the policy by acting greedily

$$\pi'(s) = \operatorname*{argmax}_{a \in \mathcal{A}} q_\pi(s, a)$$

- This improves the value from any state s over one step,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- It therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) \leq q_\pi(s, \pi'(s)) &= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s\right] \\ &\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \ldots \mid S_t = s\right] = v_{\pi'}(s) \end{aligned}$$

Policy Improvement (cont'd)

- If improvements stop,

$$q_\pi(s, \pi'(s)) = \max_{a \in \mathcal{A}} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

- Then the Bellman optimality equation has been satisfied

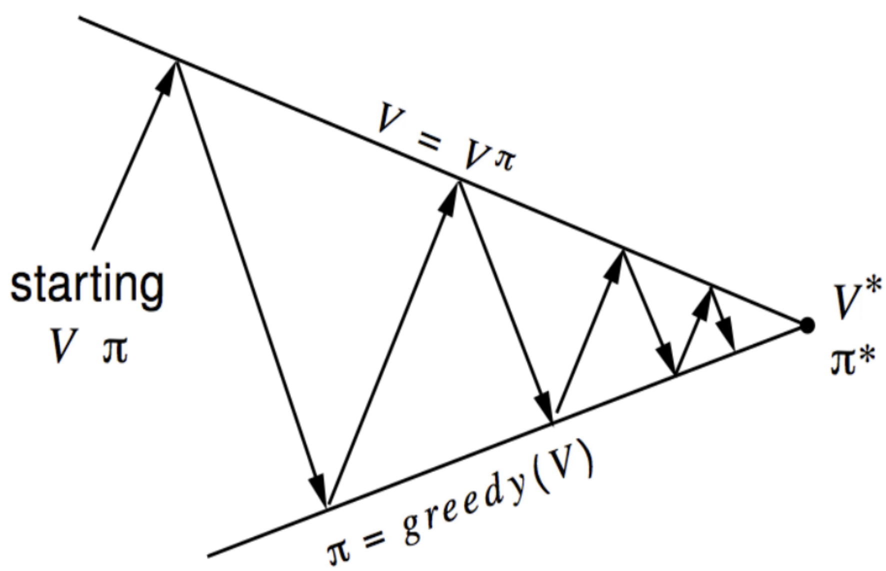
$$v_\pi(s) = \max_{a \in \mathcal{A}} q_\pi(s, a)$$

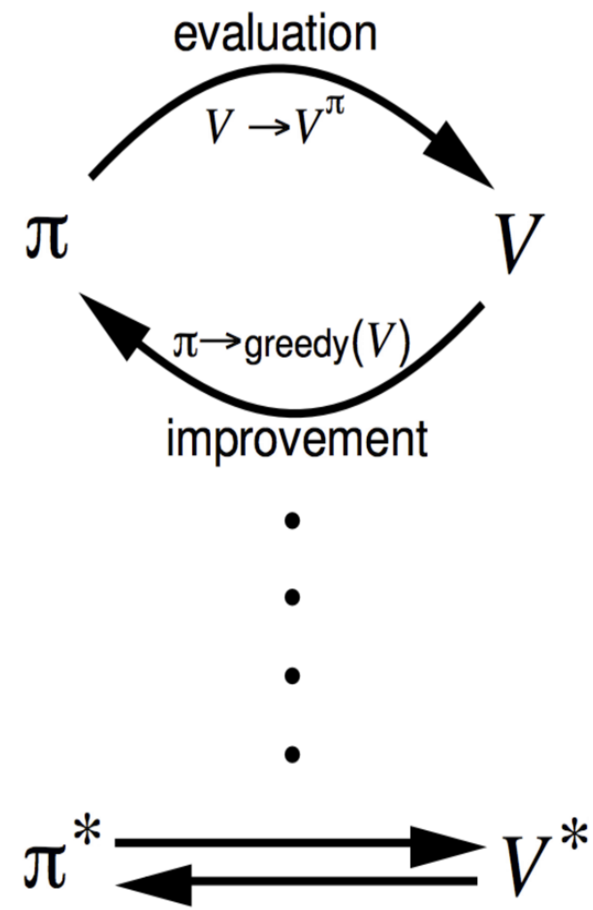
- Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$
- so π is an optimal policy

Modified Policy Iteration

- Does policy evaluation need to converge to v_π?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?
- For example, in the small gridworld $k = 3$ was sufficient to achieve optimal policy
- Why not update policy every iteration?
  - This is equivalent to *value iteration* (next section)

Generalized Policy Iteration





Policy evaluation Estimate v_π

Any policy evaluation algorithm

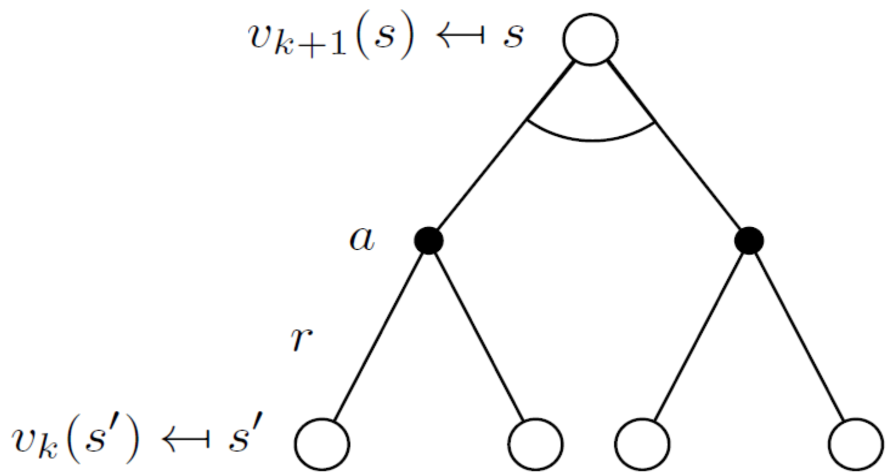
Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

Value Iteration

- Problem: find optimal policy π
- Solution: iterative application of Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_*$
- Using synchronous backups
  - At each iteration $k+1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
- Convergence to v_*
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

Value Iteration (cont'd)



$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^{\mathbf{a}} + \gamma \boldsymbol{\mathcal{P}}^{\mathbf{a}} \mathbf{v}_k$$

Complexity of Dynamic Programming Algorithms

Problem	Bellman Equation	Algorithm
Prediction	Bellman Expectation Equation	Iterative Policy Evaluation
Control	Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$
- Complexity $O(mn^2)$ per iteration, for m actions and n states
- Could also apply to action-value function $q_\pi(s,a)$ or $q_*(s,a)$
- Complexity $O(m^2n^2)$ per iteration

Question

- Comments are more than welcome!