Reinforcement Learning: A brief introduction

Mihaela van der Schaar

Outline

- **Optimal Decisions & Optimal Forecasts –**
- **Markov Decision Processes (MDPs)**
  - States, actions, rewards and value functions
  - Dynamic Programming
  - Bellman equations
    - Value Iteration
    - Policy Iteration
  - Illustrative examples
  - Reinforcement learning

Markov decision process (MDP)

- Discrete-time stochastic *control* process
- Extension of Markov chains
- Differences:
  - Addition of actions (choice)
  - Addition of rewards (goal)
- If the actions are fixed and there are no rewards, an MDP reduces to a Markov chain

Discrete MDP model

- Time t is discrete
- State space S
- Set of actions A
- Reward function $R(s,a)$
- Transition model $p(s'|s,a)$

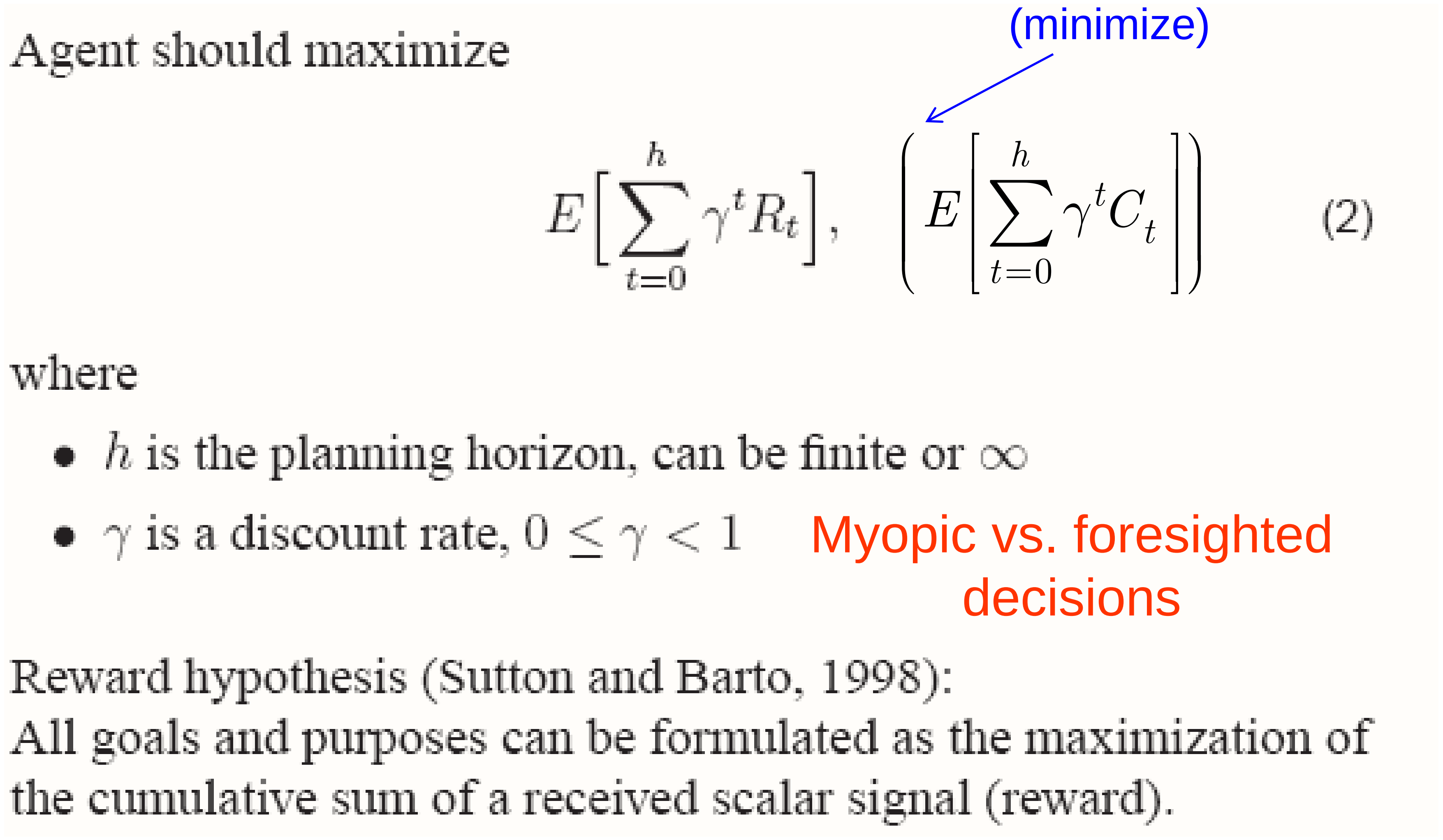
Cost function can be used instead!
$C(s,a)$

(Cost = -Reward)

The Markov property entails that the next state s_{t+1} only depends on the previous state s_t and action a_t:

$$p(s_{t+1}|s_t, s_{t-1}, \dots, s_0, a_t, a_{t-1}, \dots, a_0) = p(s_{t+1}|s_t, a_t)$$

Rewards and optimality criterion



Agent should maximize

(minimize)

$$E\left[\sum_{t=0}^{h}\gamma^t R_t\right], \quad \left(E\left[\sum_{t=0}^{h}\gamma^t C_t\right]\right) \qquad (2)$$

where

- h is the planning horizon, can be finite or ∞
- γ is a discount rate, $0 \leq \gamma < 1$

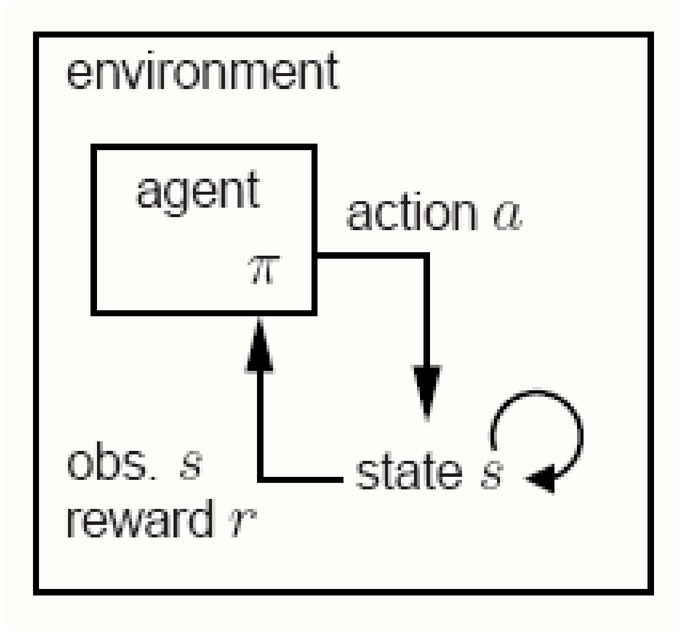
Myopic vs. foresighted decisions

Reward hypothesis (Sutton and Barto, 1998):
All goals and purposes can be formulated as the maximization of the cumulative sum of a received scalar signal (reward).

Examples?

Solution to an MDP = Policy π

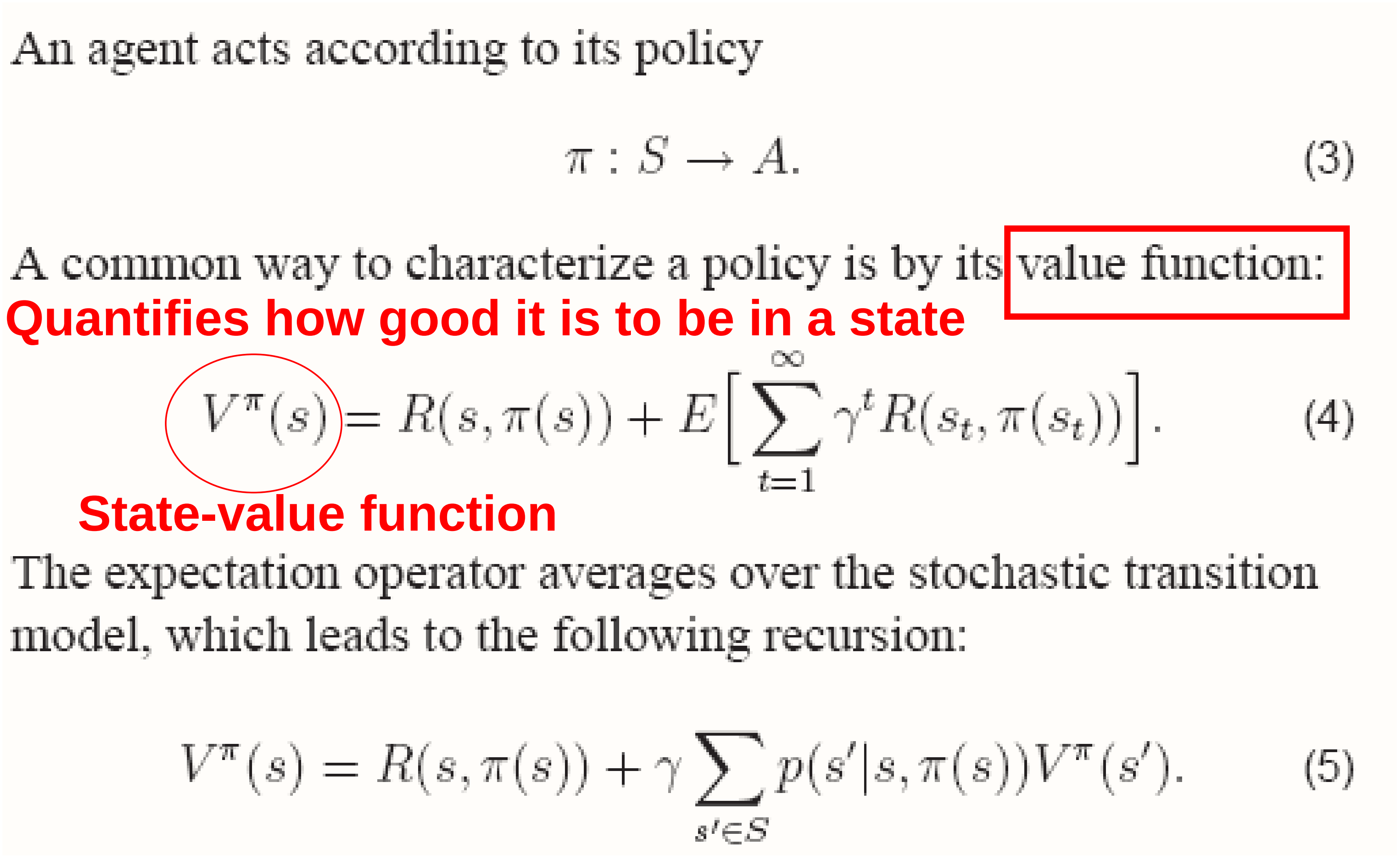
- Gives the action to take from a given state.
- Does not depend on history (Why?)
- Goal: Find a policy that maximizes the cumulative discounted sum of rewards $\left(E\left[\sum_{t=0}^{h} \gamma^t R_t \right] \right)$



Brief summary of concepts

- The *agent* and its *environment* interact over a sequence of discrete time steps.
- The specification of their interface defines a particular task:
  - the *actions* are the choices made by the agent;
  - the *states* are the basis for making the choices;
  - the *rewards* are the basis for evaluating the choices.
- A *policy* is a stochastic rule by which the agent selects actions as a function of states.
- The agent's objective is to maximize the amount of reward it receives over time.

Policies and value



An agent acts according to its policy

$$\pi : S \rightarrow A. \tag{3}$$

A common way to characterize a policy is by its value function:

Quantifies how good it is to be in a state

$$V^{\pi}(s) = R(s, \pi(s)) + E\Big[\sum_{t=1}^{\infty} \gamma^t R(s_t, \pi(s_t))\Big]. \tag{4}$$

State-value function

The expectation operator averages over the stochastic transition model, which leads to the following recursion:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^{\pi}(s'). \tag{5}$$

Policies and value

Extracting a policy π from a value function V is easy:

$$\pi(s) = \arg\max_{a\in A} \Big[R(s,a) + \gamma \sum_{s'\in S} p(s'|s,a) V(s') \Big]. \qquad (6)$$

Bellman (1957) equation:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$V^*(s) = \max_{a\in A} \Big[R(s,a) + \gamma \sum_{s'\in S} p(s'|s,a) V^*(s') \Big], \qquad (7)$$

Optimal state-value function

Known

Solving this (nonlinear) system of equations for each state s yields the optimal value function, and an optimal policy π^*. However, due to the nonlinear max operator solving the system for each state simultaneously is not efficient for large MDPs

Brief summary of concepts

- A policy's *value function* assigns to each state the expected return from that state given that the agent uses the policy.
- The *optimal value function* assigns to each state the largest expected return achievable by any policy.
- A policy whose value functions are optimal is an *optimal policy*.
- The optimal value functions for states are unique for a given MDP, but there can be many optimal policies.
- The *Bellman optimality equations* are special consistency conditions that the optimal value functions must satisfy and that can, in principle, be solved to obtain the optimal value functions, from which an optimal policy can be determined with relative ease.

How do we compute the optimal state-value function and the optimal policy?

- Problem:
  - Given:
    - Transition probability function: $p\left(s' \mid s, a\right)$
    - Reward function: $R(s,a)$
  - Determine:
    - Optimal state-value function: V^*
    - Optimal policy: π^*

Solution:

Dynamic Programming

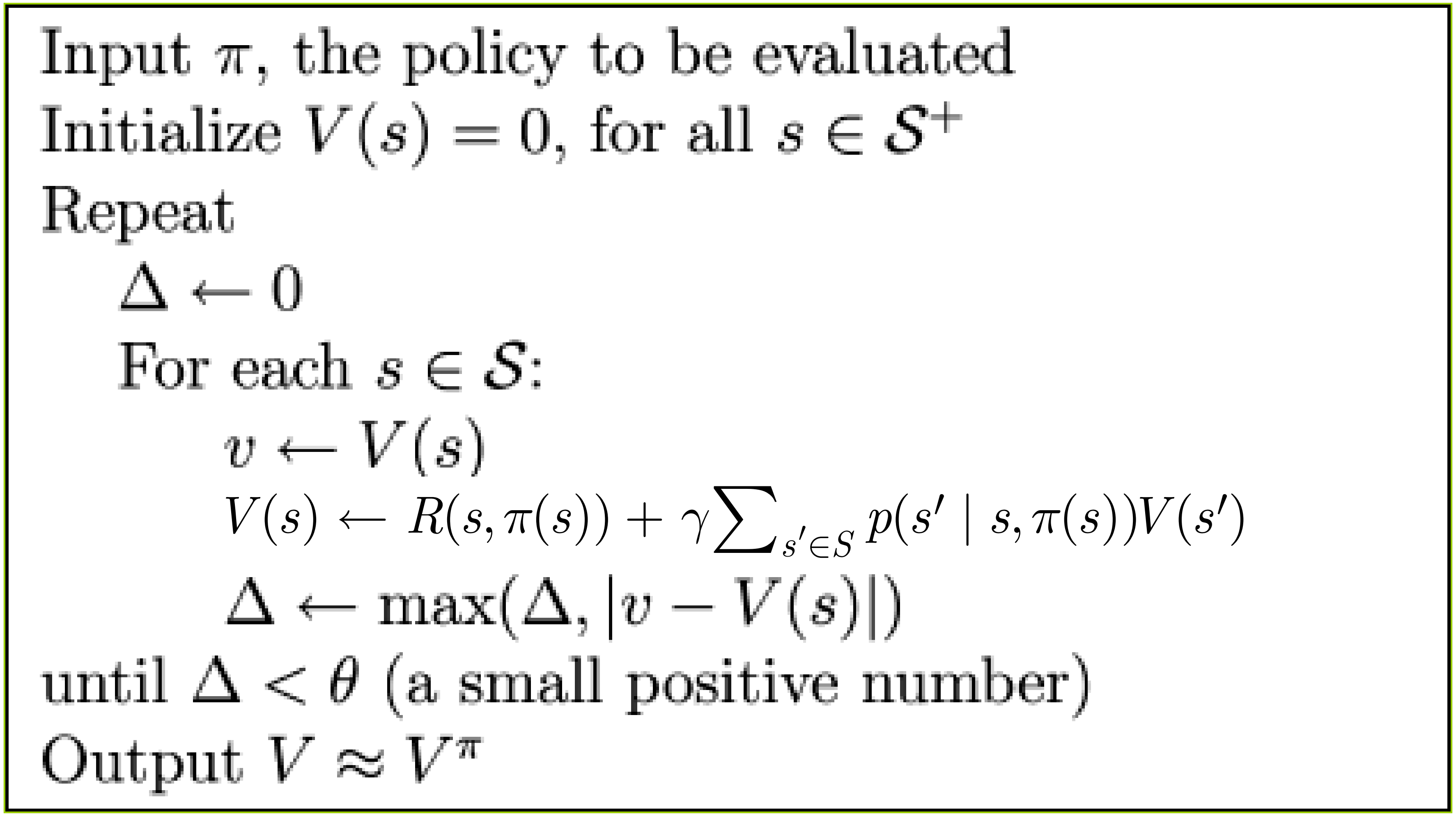
Dynamic Programming

- The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment
  - Can be used to solve Markov decision processes.
  - Use value functions to organize and structure the search for good policies.
  - Turn Bellman equations into update policies.

DP methods

- *Policy evaluation* refers to the (typically) iterative computation of the value functions for a given policy.
- *Policy improvement* refers to the computation of an improved policy given the value function for that policy.
- Putting these two computations together, we obtain *policy iteration* and *value iteration*, the two most popular DP methods.
  - Either of these can be used to reliably compute optimal policies and value functions for finite MDPs given complete knowledge of the MDP.

Policy evaluation



```
Input $\pi$, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s' \in S} p(s' \mid s, \pi(s)) V(s')$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$
```

Policy improvement

- When should we change the policy?
  - Let π be the current policy
  - Let $\pi' = \pi$ except in state s where we let $\pi'(s) = \alpha$
  - If we pick a new action α to take in state s, and $V(\pi') \geq V(\pi)$, then picking α from state s is a better policy overall.

- The above follows from the *policy improvement theorem*

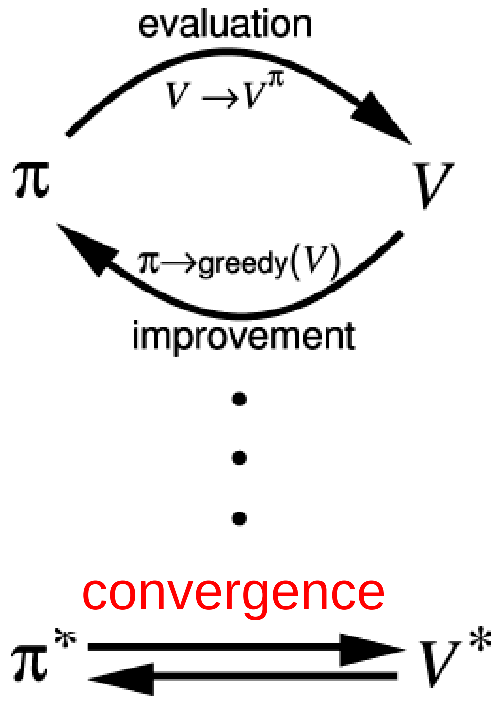
Iterative DP algorithms

- All iterative algorithms for computing the optimal state-value function and optimal policy have two basic steps:
  - 1. Policy Improvement
  - 2. Policy Evaluation

1. $$\pi(s) := \arg\max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right\}$$

2. $$V(s) := R(s,\pi(s)) + \gamma \sum_{s' \in S} p(s' \mid s,\pi(s)) V(s')$$

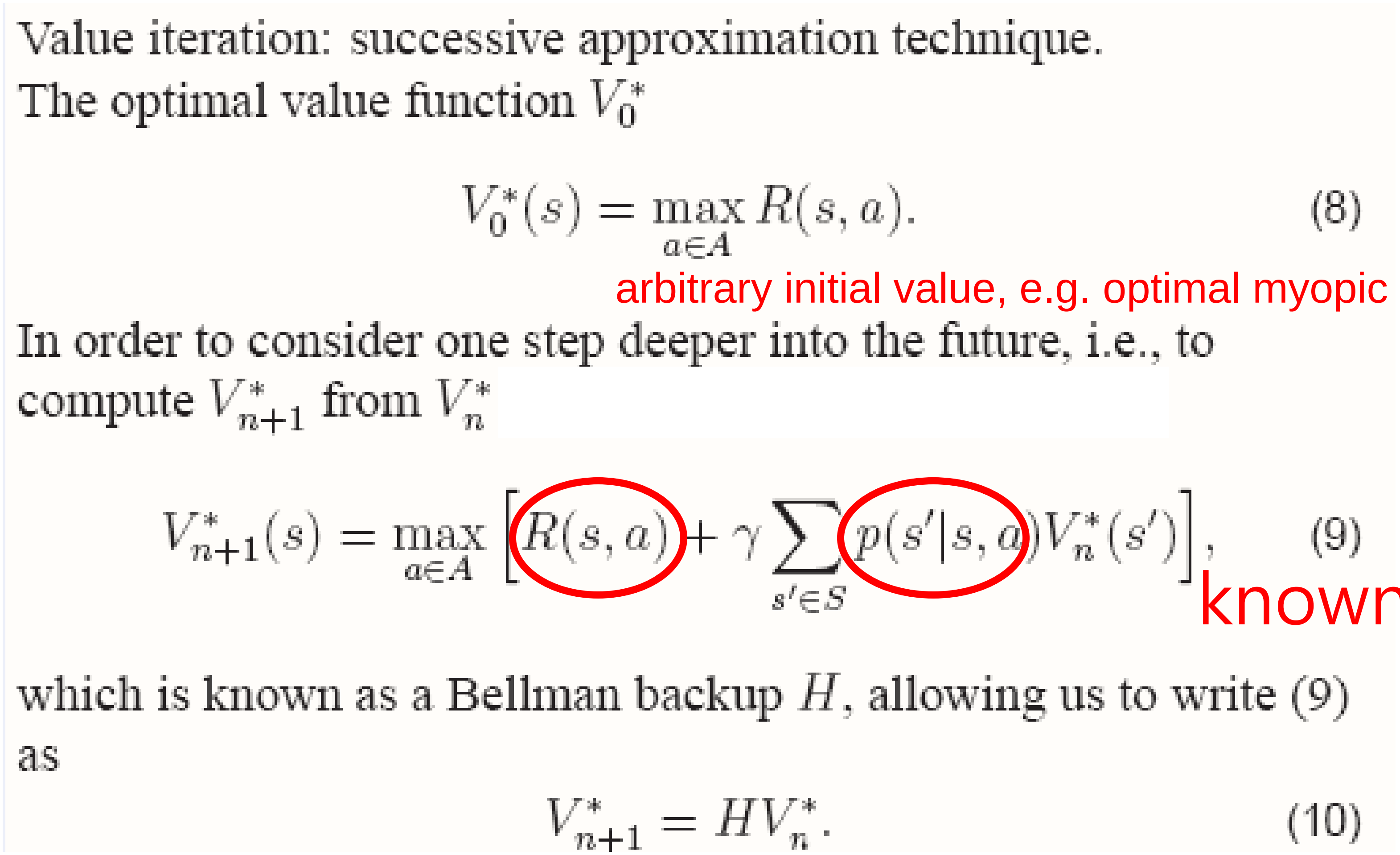
Value Function



Next, several algorithms that apply these two basic steps (in different orders) for computing optimal value function and optimal policy

Value iteration

can be used to compute optimal policies and value functions



Value iteration: successive approximation technique.
The optimal value function V_0^*

$$V_0^*(s) = \max_{a \in A} R(s, a). \tag{8}$$

arbitrary initial value, e.g. optimal myopic value

In order to consider one step deeper into the future, i.e., to compute V_{n+1}^* from V_n^*

$$V_{n+1}^*(s) = \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} p(s' | s, a) V_n^*(s') \right], \tag{9}$$

known

which is known as a Bellman backup H, allowing us to write (9) as

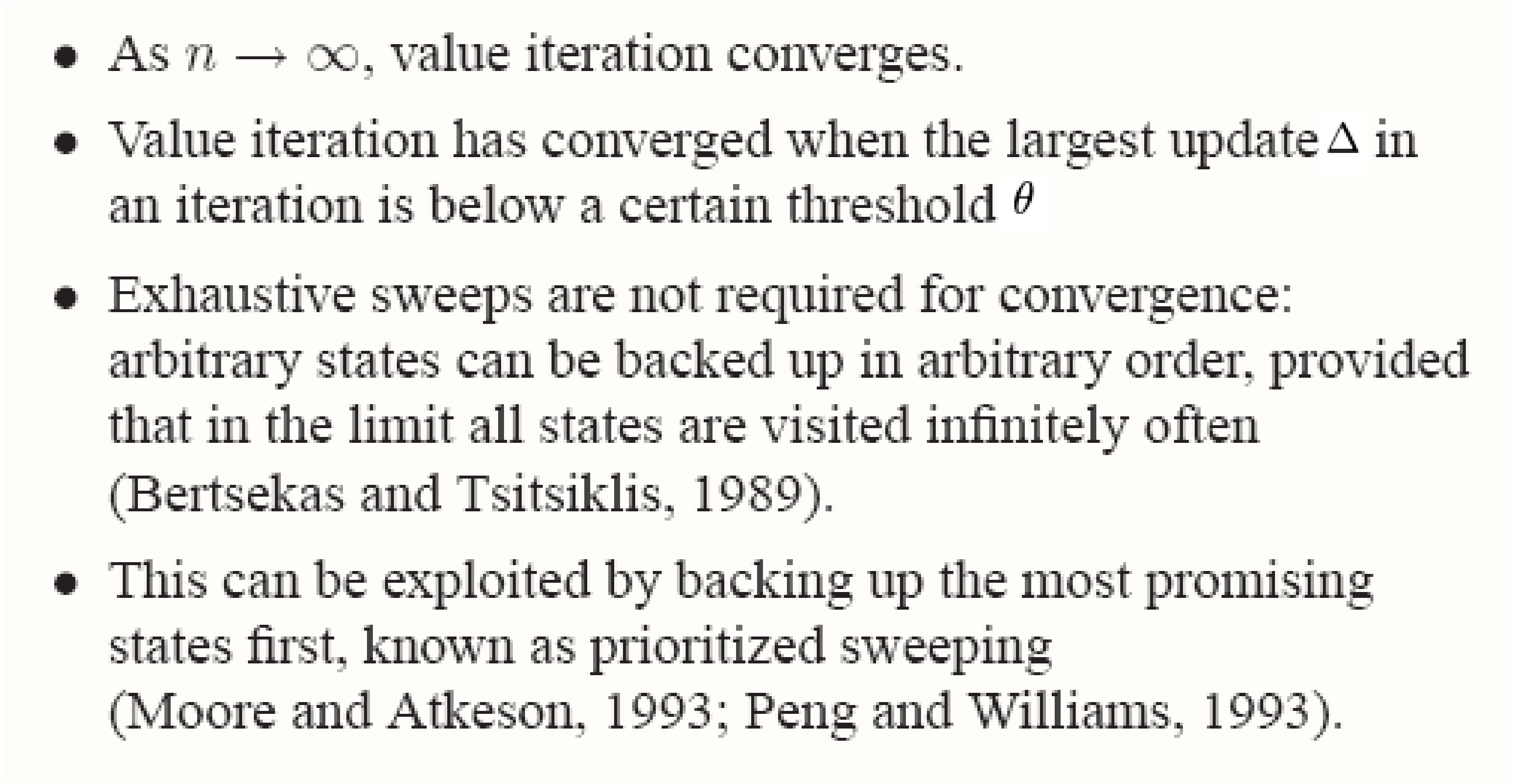
$$V_{n+1}^* = H V_n^*. \tag{10}$$

Note that policy improvement and evaluation in (9) are combined (simultaneous)

Value iteration

```
Initialize $V$ arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right\}$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
  $\pi(s) \leftarrow \arg\max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right\}$
```

Value iteration discussion

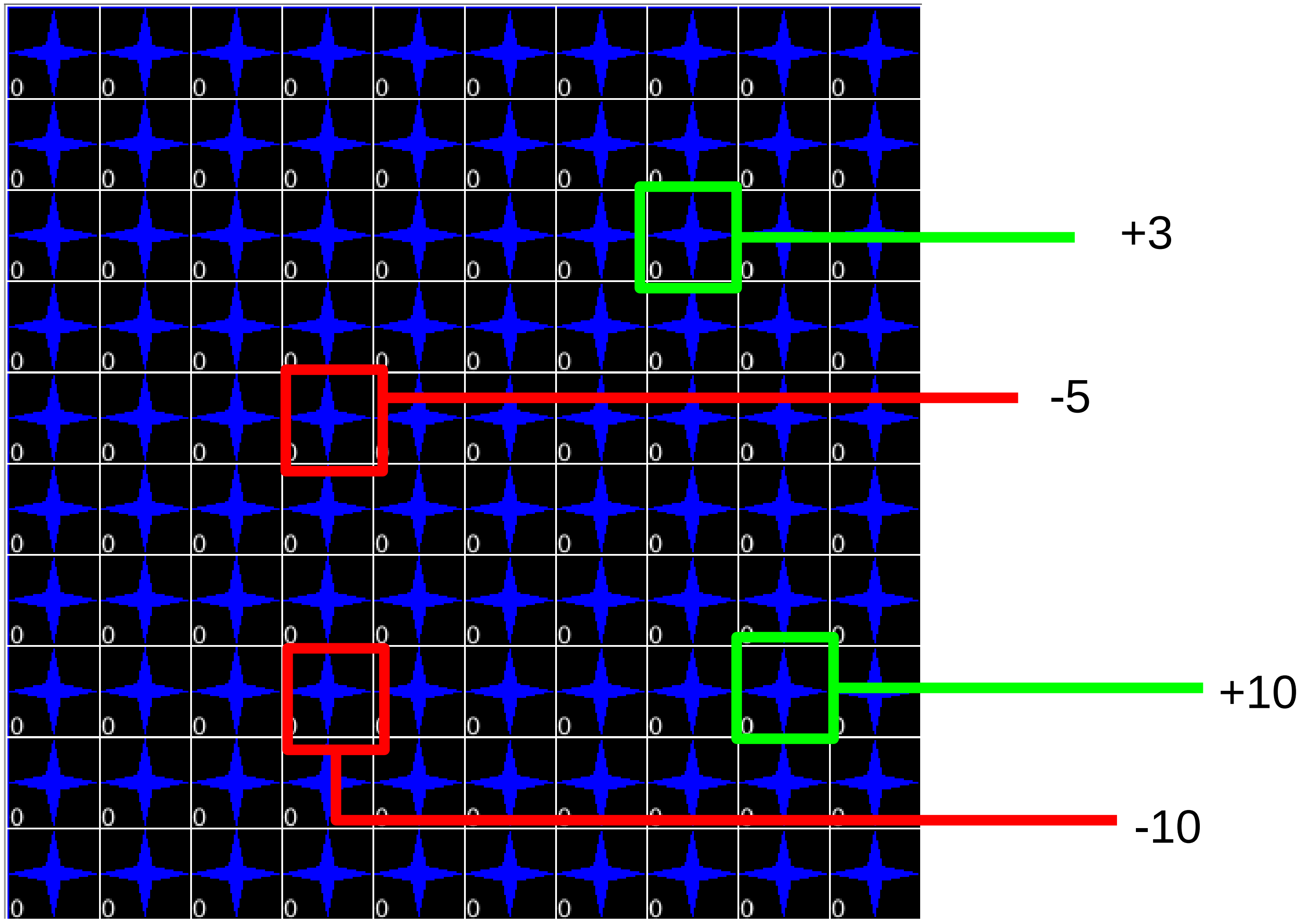


- As $n \to \infty$, value iteration converges.
- Value iteration has converged when the largest update Δ in an iteration is below a certain threshold θ
- Exhaustive sweeps are not required for convergence: arbitrary states can be backed up in arbitrary order, provided that in the limit all states are visited infinitely often (Bertsekas and Tsitsiklis, 1989).
- This can be exploited by backing up the most promising states first, known as prioritized sweeping (Moore and Atkeson, 1993; Peng and Williams, 1993).

Value iteration example: Grid World

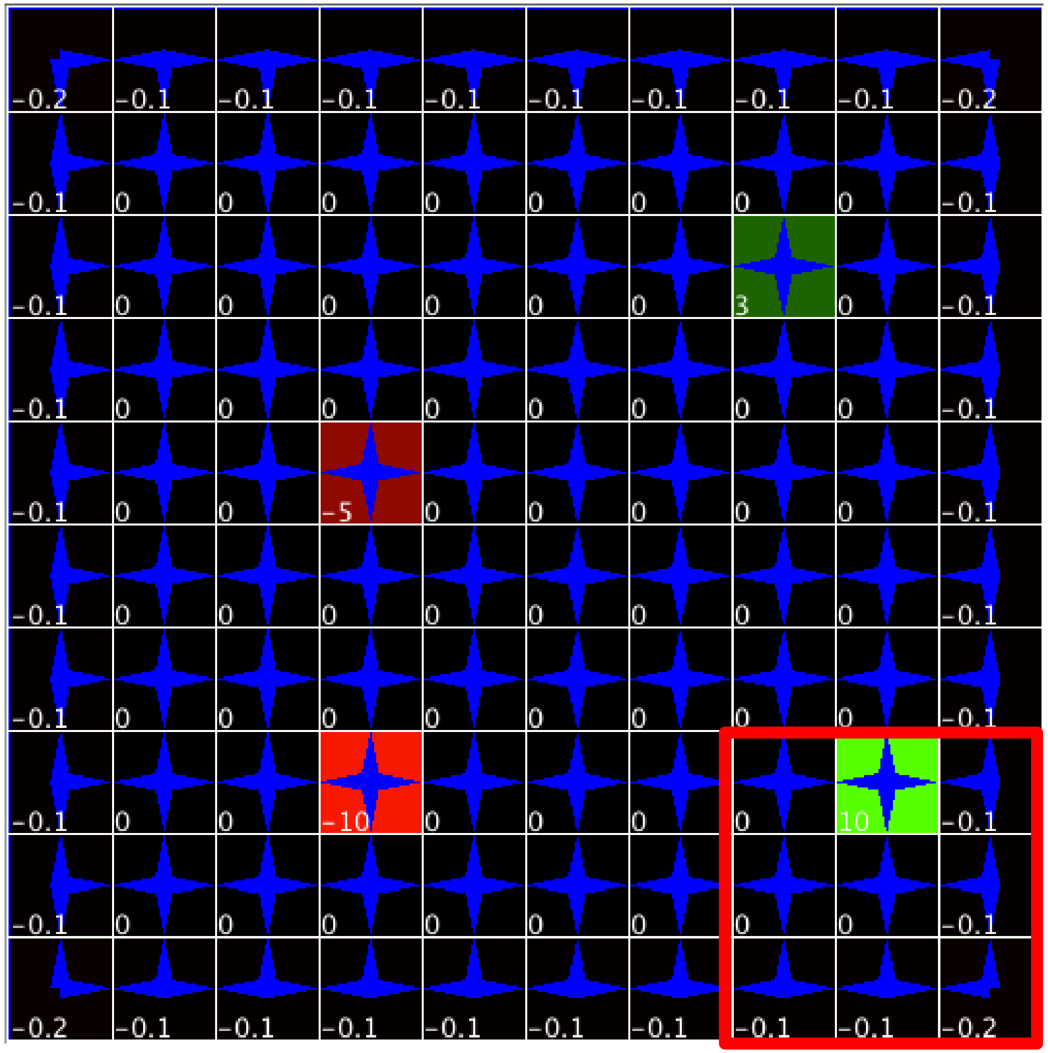
- States: (x,y)-coordinate in a 10 x 10 grid
- Actions: Up, down, left, and right
- Dynamics:
  - Move one-step in direction specified by action with prob 0.7
  - Move one-step in any one of the other directions with prob 0.1
  - Cannot move outside of the grid (i.e. end up in the same state)
  - Agent is flung randomly to corner of grid after entering a Goal or Penalty state
- Rewards:
  - Attempted move outside of grid leads to reward of -1
  - Goal/Penalty states in grid: +10@(9,8), +3@(8,3), -5@(4,5), -10@(4,8)
  - No reward in any other states
- Discount factor: 0.9

Value iteration example: Grid World



10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World



-0.2	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.2
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.1	0	0	0	0	0	0	3	0	-0.1
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.1	0	0	-5	0	0	0	0	0	-0.1
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.1	0	0	-10	0	0	0	0	10	-0.1
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.2	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.2

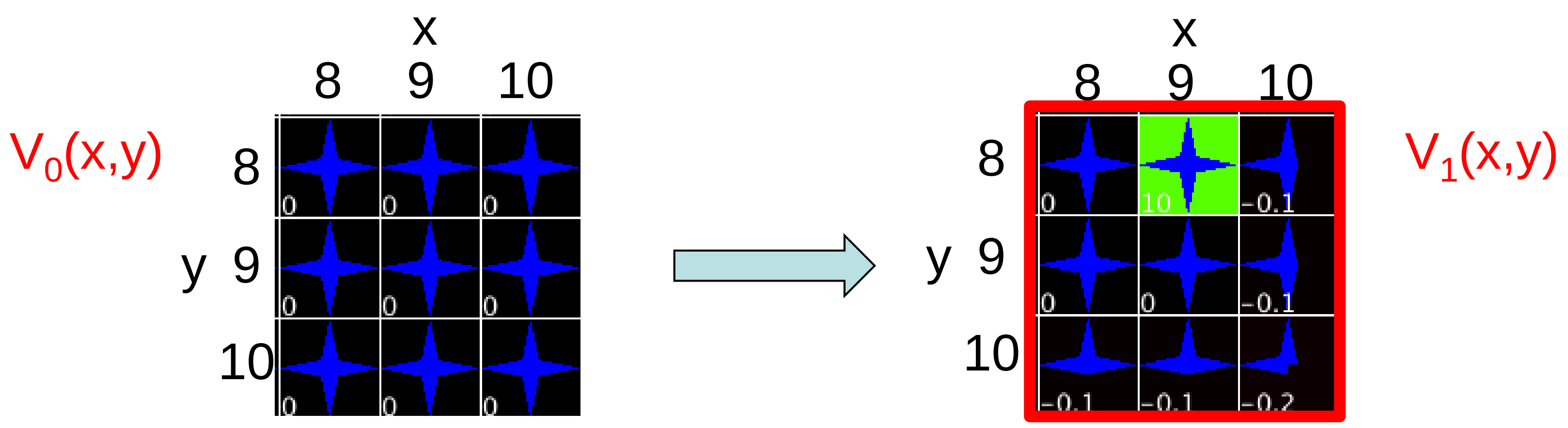
$V_1(x,y)$ (Iteration 1)

How are these calculated?

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World

$$
\begin{aligned}
V_1^*\left(9,8\right) &= \max_a \left[R\left(9,8,a\right) + \gamma \sum_{\left(x',y'\right)} p\left(x',y' \mid 9,8,a\right) V_0^*\left(x',y'\right) \right] \\
&= \max_a R\left(9,8,a\right) \\
&= \max_a \left\{ \underbrace{10}_{\text{up}}, \underbrace{10}_{\text{down}}, \underbrace{10}_{\text{left}}, \underbrace{10}_{\text{right}} \right\} = 10
\end{aligned}
$$



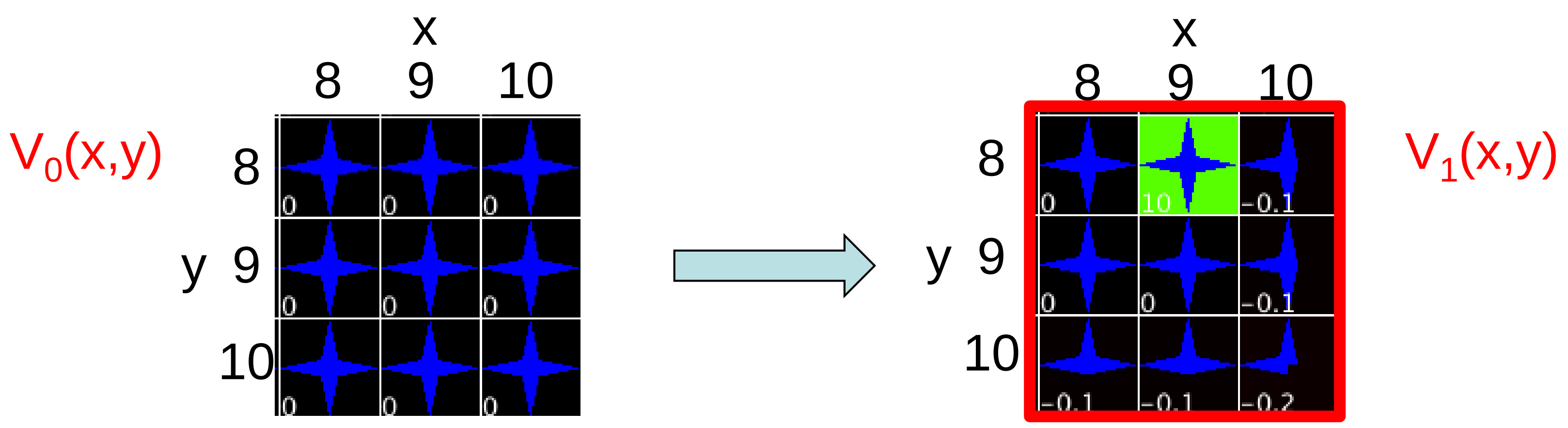
10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World

$$V_1^*(10,10) = \max_a \left[R(10,10,a) + \gamma \sum_{(x',y')} p(x',y' \mid 10,10,a) V_0^*(x',y') \right]$$

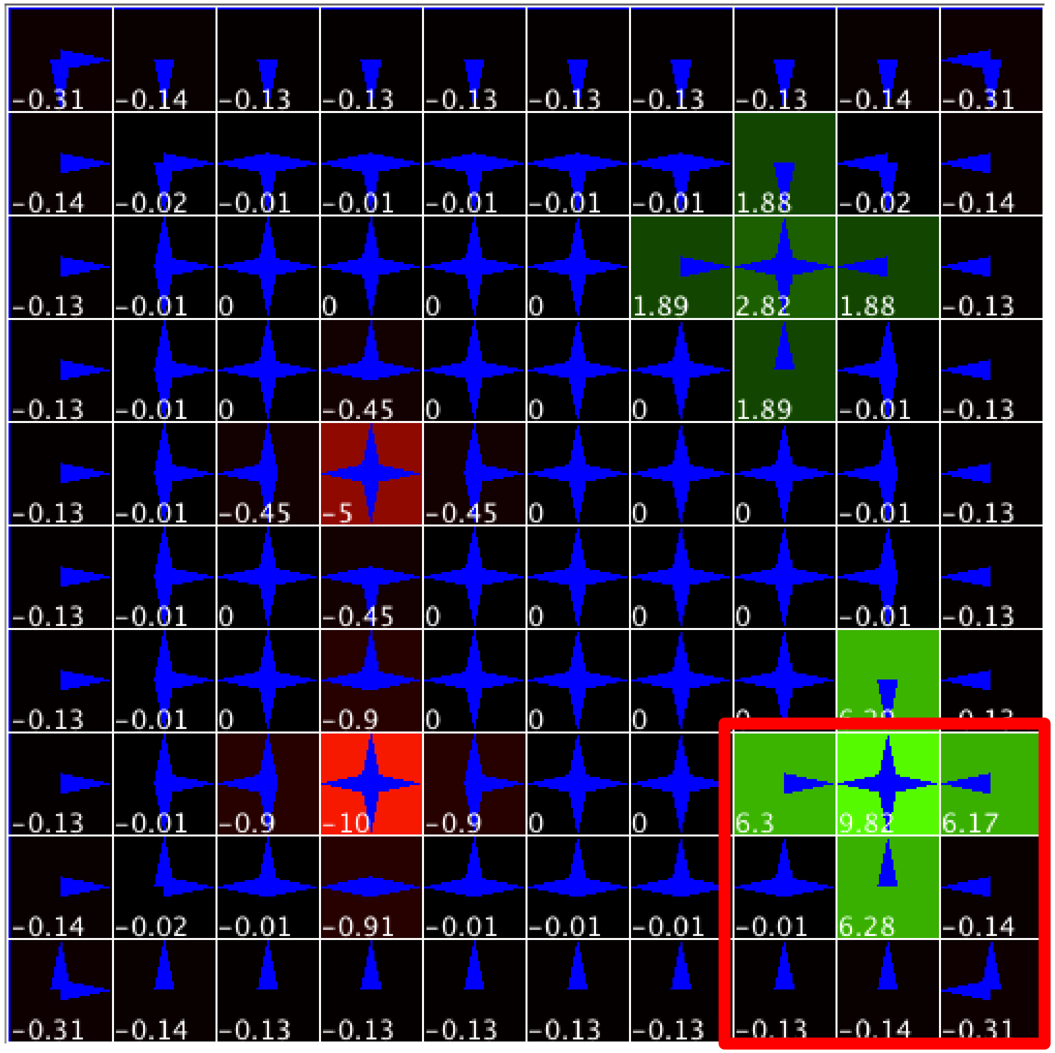
$$= \max_a R(10,10,a)$$

$$= \max_a \left\{ \underbrace{2\cdot 0.1(-1)}_{\text{up}}, \underbrace{(0.7+0.1)(-1)}_{\text{down}}, \underbrace{2\cdot 0.1(-1)}_{\text{left}}, \underbrace{(0.7+0.1)(-1)}_{\text{right}} \right\} = -0.2$$



10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World



-0.31	-0.14	-0.13	-0.13	-0.13	-0.13	-0.13	-0.13	-0.14	-0.31
-0.14	-0.02	-0.01	-0.01	-0.01	-0.01	-0.01	1.88	-0.02	-0.14
-0.13	-0.01	0	0	0	0	1.89	2.82	1.88	-0.13
-0.13	-0.01	0	-0.45	0	0	0	1.89	-0.01	-0.13
-0.13	-0.01	-0.45	-5	-0.45	0	0	0	-0.01	-0.13
-0.13	-0.01	0	-0.45	0	0	0	0	-0.01	-0.13
-0.13	-0.01	0	-0.9	0	0	0	0	6.28	-0.13
-0.13	-0.01	-0.9	-10	-0.9	0	0	6.3	9.82	6.17
-0.14	-0.02	-0.01	-0.91	-0.01	-0.01	-0.01	-0.01	6.28	-0.14
-0.31	-0.14	-0.13	-0.13	-0.13	-0.13	-0.13	-0.13	-0.14	-0.31

$V_2(x,y)$ (Iteration 2)

How are these calculated?

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

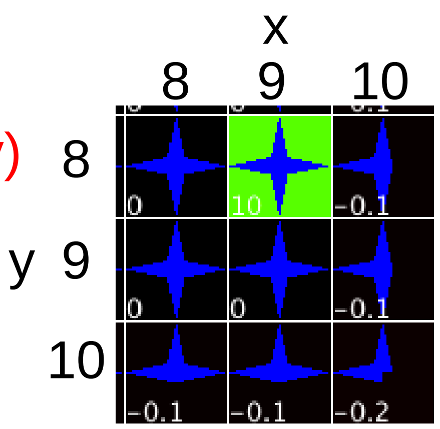
Value iteration example: Grid World

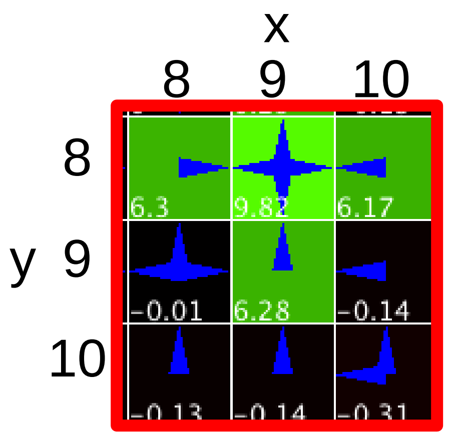
$$V_2^*(9,8) = \max_a \left[R(9,8,a) + \gamma \sum_{(x',y')} p(x',y' \mid 9,8,a) V_1^*(x',y') \right]$$

$$= 10 + 0.9 \times 0.25 \left[V(1,1) + V(1,10) + V(10,1) + V(10,10) \right]$$

$$= 10 + 0.9 \times -0.2 = 9.82$$

$V_1(x,y)$





$V_2(9,8)$

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

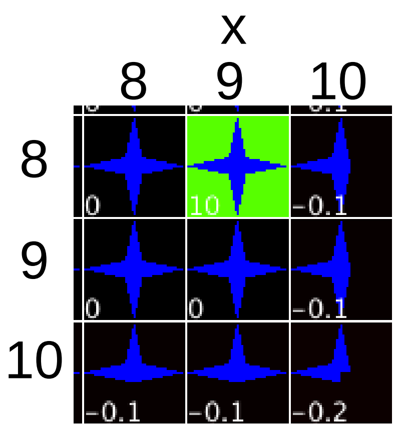
Value iteration example: Grid World

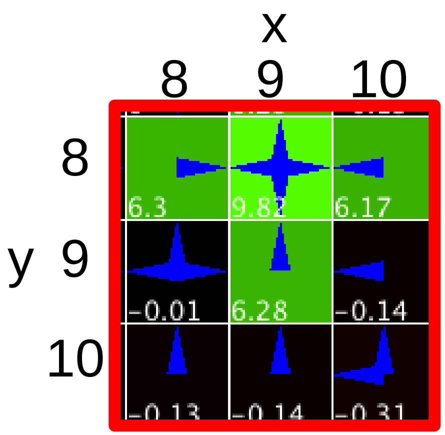
$$V_2^*(10,10) = \max_a \left[R(10,10,a) + \gamma \sum_{(x',y')} p(x',y' \mid 10,10,a) V_1^*(x',y') \right]$$

$$a = \text{up} \qquad -0.2 + 0.9\big[0.7(-0.1) + 0.1(-0.1) + 2\cdot 0.1(-0.2)\big] = -0.31$$

$$a = \text{down} \qquad -0.8 + 0.9\big[(0.7 + 0.1)(-0.2) + 2\cdot 0.1(-0.1)\big] = -0.96$$

$V_1(x,y)$





$V_2(10,10)$

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

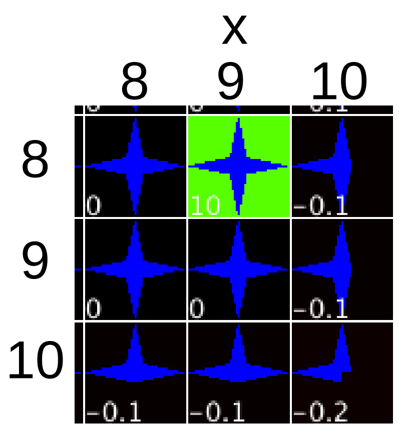
Value iteration example: Grid World

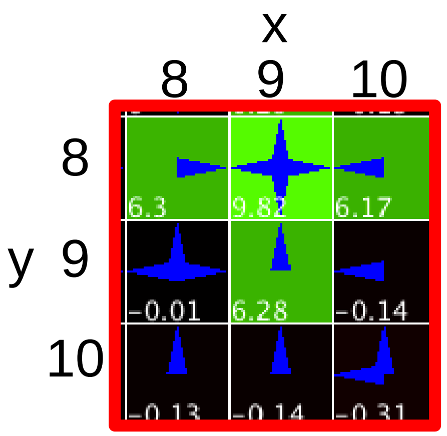
$$V_2^*(10,10) = \max_a \left[R(10,10,a) + \gamma \sum_{(x',y')} p(x',y' \mid 10,10,a) V_1^*(x',y') \right]$$

$$a = \text{left} \qquad -0.2 + 0.9\left[0.7(-0.1) + 0.1(-0.1) + 2 \cdot 0.1(-0.2)\right] = -0.31$$

$$a = \text{right} \qquad -0.8 + 0.9\left[(0.7 + 0.1)(-0.2) + 2 \cdot 0.1(-0.1)\right] = -0.96$$

$V_1(x,y)$





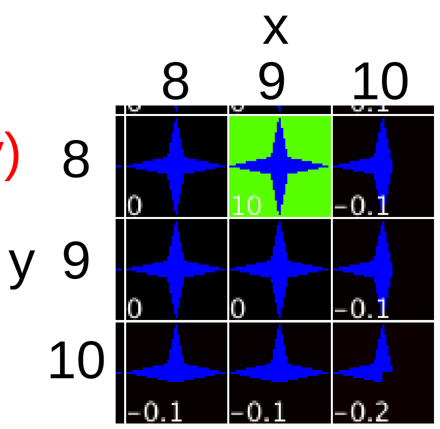
$V_2(10,10)$

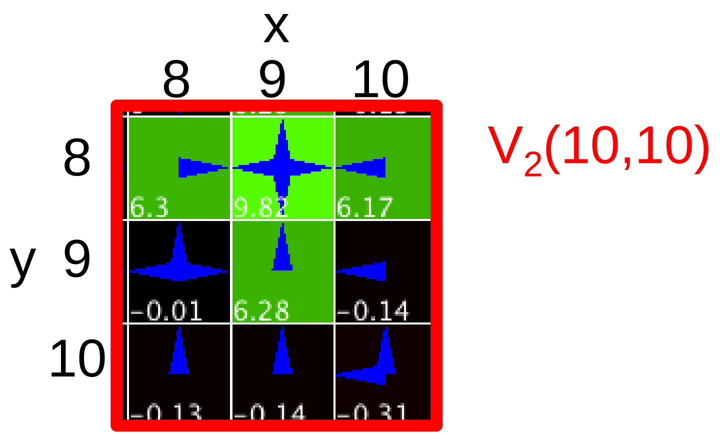
10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World

$$V_2^*(10,10) = \max_a \left[R(10,10,a) + \gamma \sum_{(x',y')} p(x',y' \mid 10,10,a) V_1^*(x',y') \right]$$

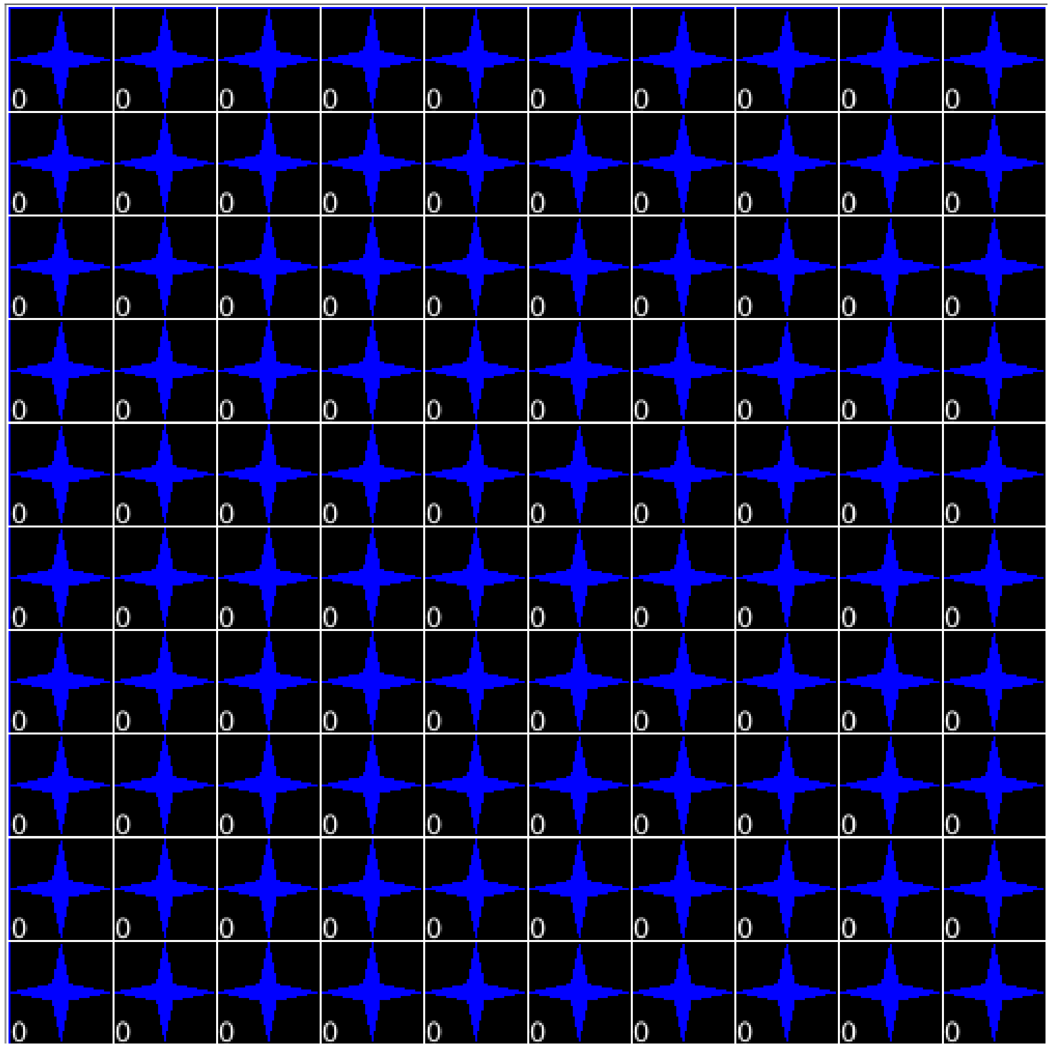
$$= -0.31$$





10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World

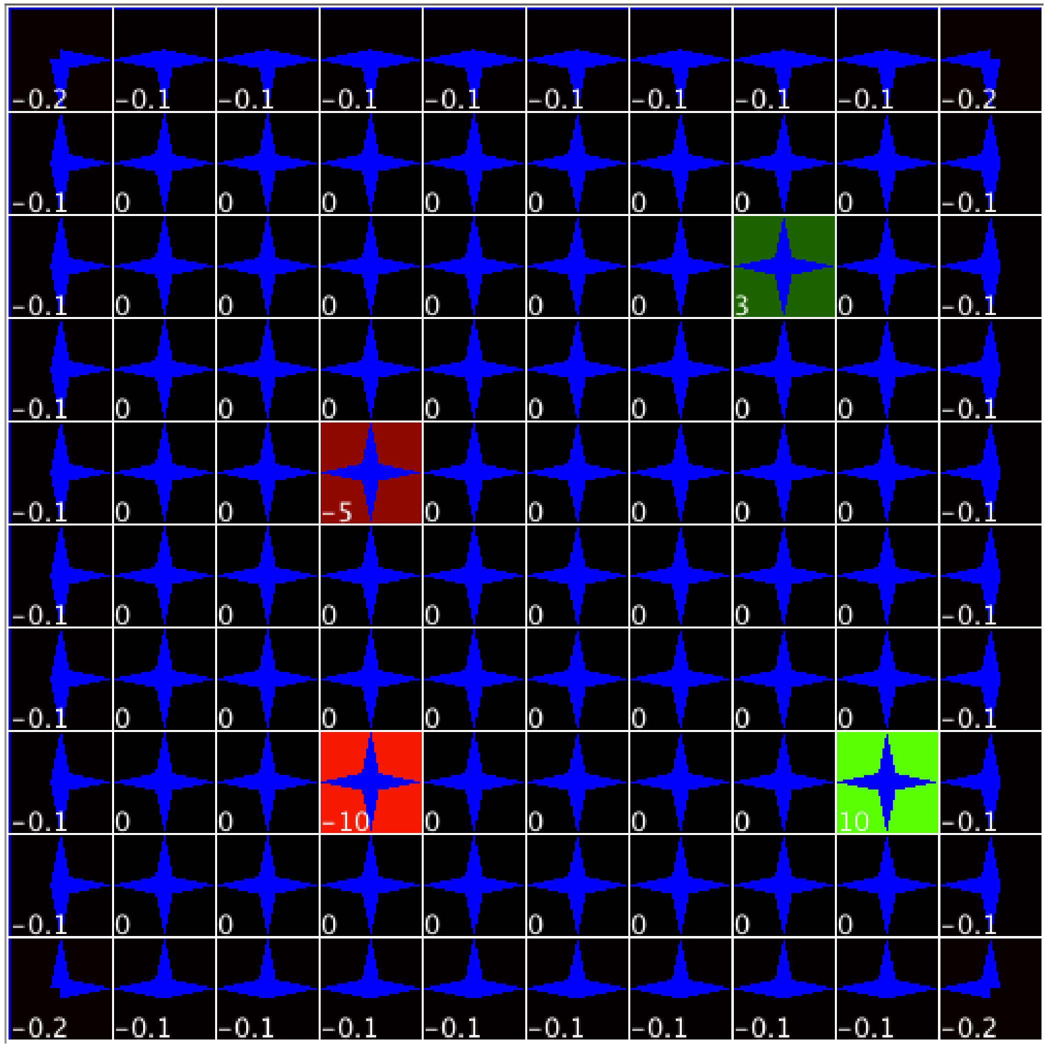


0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$V_0(x,y)$ (Iteration 0)

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World

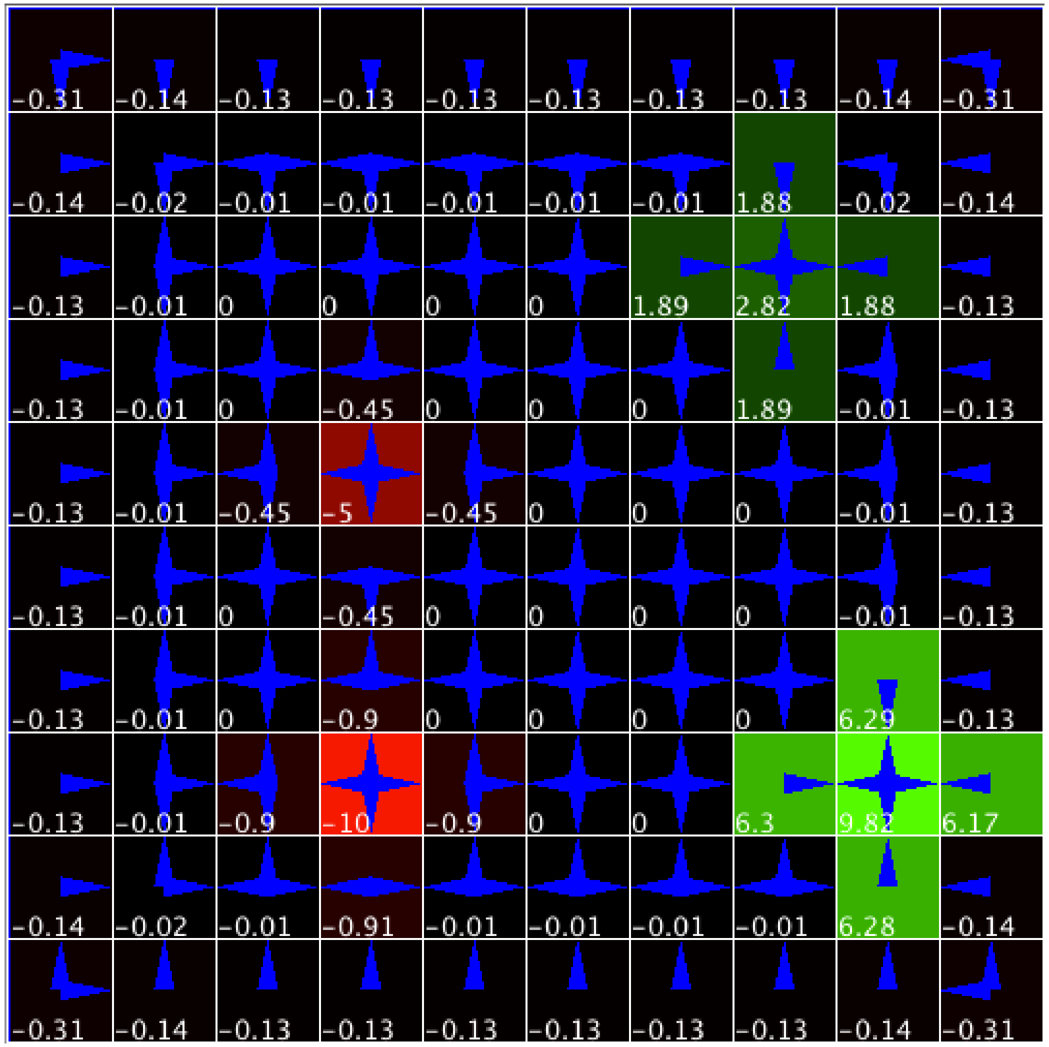


-0.2	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.2
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.1	0	0	0	0	0	0	3	0	-0.1
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.1	0	0	-5	0	0	0	0	0	-0.1
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.1	0	0	-10	0	0	0	0	10	-0.1
-0.1	0	0	0	0	0	0	0	0	-0.1
-0.2	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.1	-0.2

$V_1(x,y)$ (Iteration 1)

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World

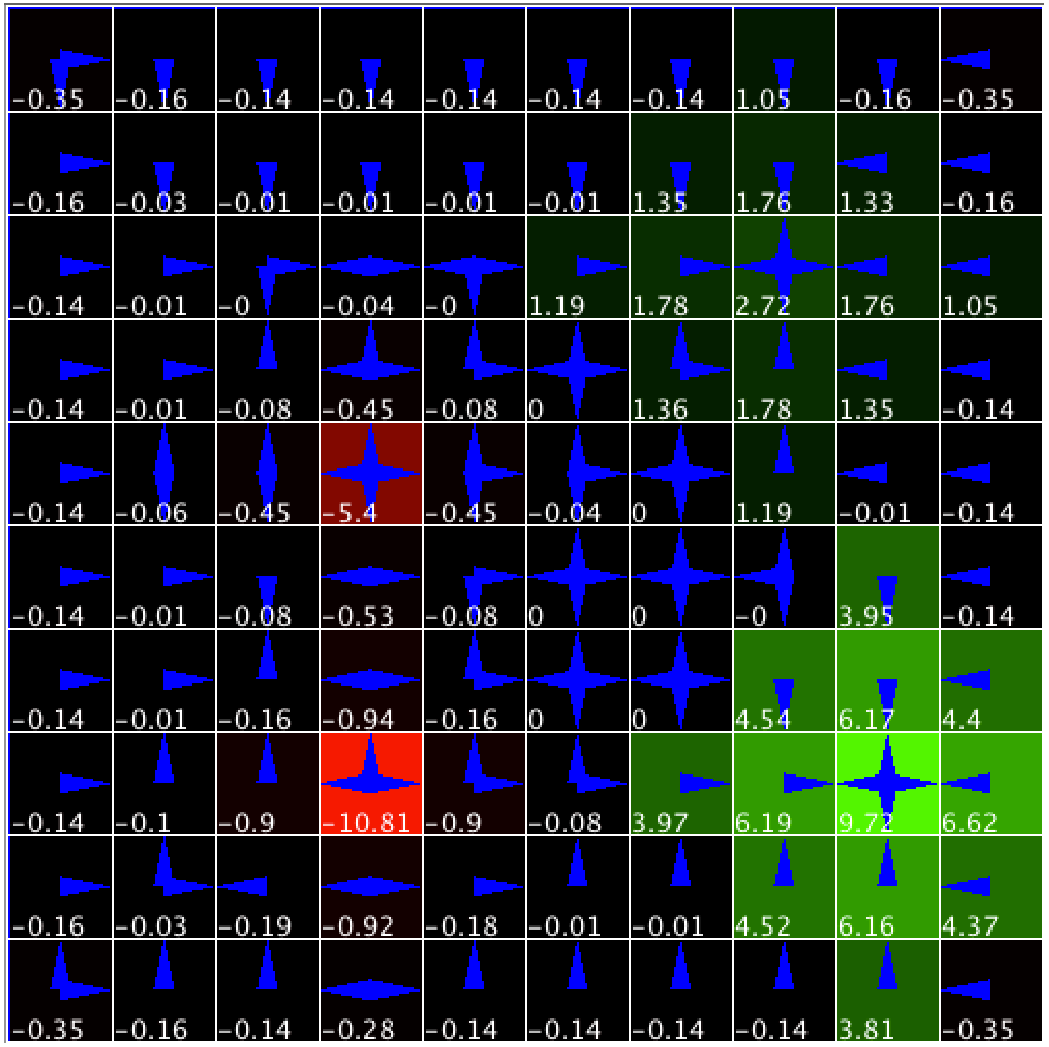


-0.31	-0.14	-0.13	-0.13	-0.13	-0.13	-0.13	-0.13	-0.14	-0.31
-0.14	-0.02	-0.01	-0.01	-0.01	-0.01	-0.01	1.88	-0.02	-0.14
-0.13	-0.01	0	0	0	0	1.89	2.82	1.88	-0.13
-0.13	-0.01	0	-0.45	0	0	0	1.89	-0.01	-0.13
-0.13	-0.01	-0.45	-5	-0.45	0	0	0	-0.01	-0.13
-0.13	-0.01	0	-0.45	0	0	0	0	-0.01	-0.13
-0.13	-0.01	0	-0.9	0	0	0	0	6.29	-0.13
-0.13	-0.01	-0.9	-10	-0.9	0	0	6.3	9.82	6.17
-0.14	-0.02	-0.01	-0.91	-0.01	-0.01	-0.01	-0.01	6.28	-0.14
-0.31	-0.14	-0.13	-0.13	-0.13	-0.13	-0.13	-0.13	-0.14	-0.31

$V_2(x,y)$ (Iteration 2)

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World

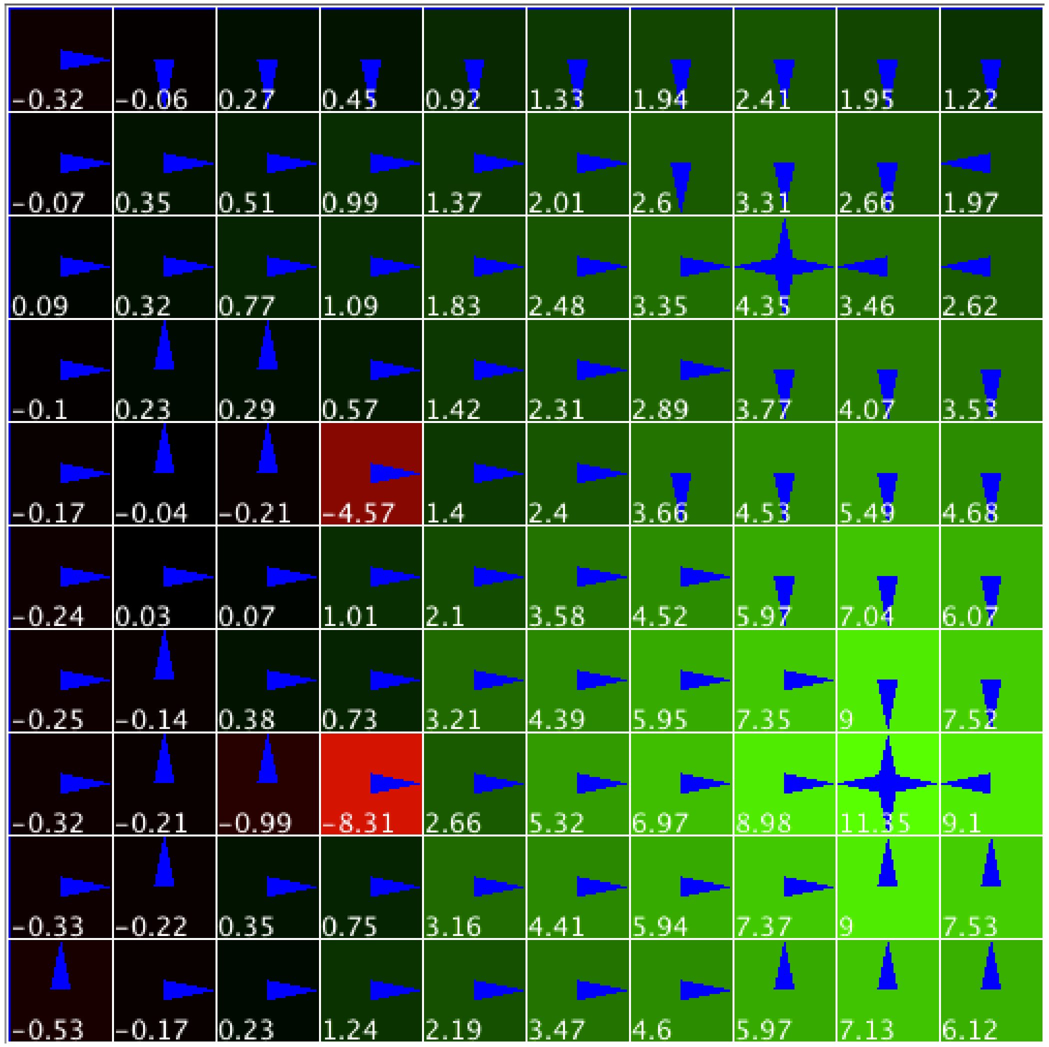


-0.35	-0.16	-0.14	-0.14	-0.14	-0.14	-0.14	1.05	-0.16	-0.35
-0.16	-0.03	-0.01	-0.01	-0.01	-0.01	1.35	1.76	1.33	-0.16
-0.14	-0.01	-0	-0.04	-0	1.19	1.78	2.72	1.76	1.05
-0.14	-0.01	-0.08	-0.45	-0.08	0	1.36	1.78	1.35	-0.14
-0.14	-0.06	-0.45	-5.4	-0.45	-0.04	0	1.19	-0.01	-0.14
-0.14	-0.01	-0.08	-0.53	-0.08	0	0	-0	3.95	-0.14
-0.14	-0.01	-0.16	-0.94	-0.16	0	0	4.54	6.17	4.4
-0.14	-0.1	-0.9	-10.81	-0.9	-0.08	3.97	6.19	9.72	6.62
-0.16	-0.03	-0.19	-0.92	-0.18	-0.01	-0.01	4.52	6.16	4.37
-0.35	-0.16	-0.14	-0.28	-0.14	-0.14	-0.14	-0.14	3.81	-0.35

$V_2(x,y)$ (Iteration 3)

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World

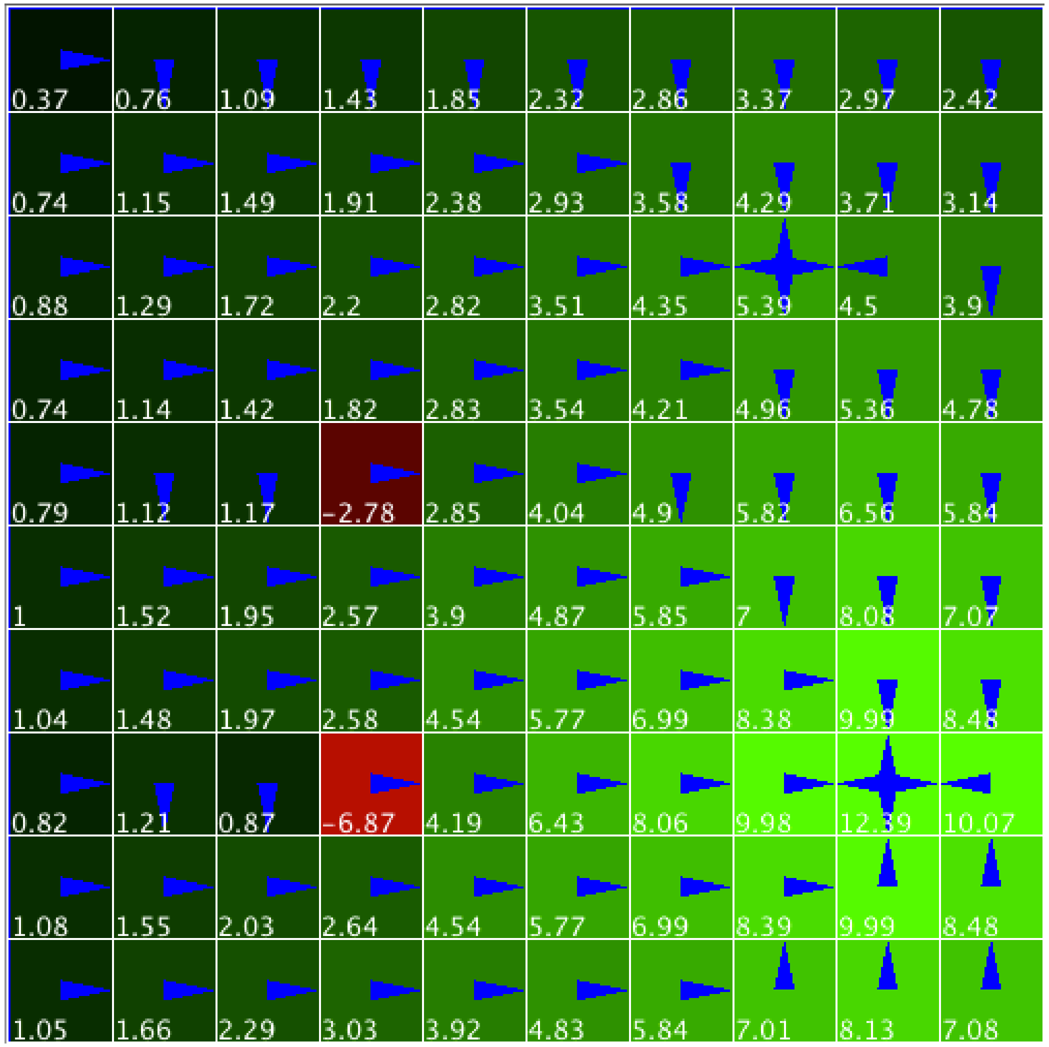


→ -0.32	↓ -0.06	↓ 0.27	↓ 0.45	↓ 0.92	↓ 1.33	↓ 1.94	↓ 2.41	↓ 1.95	↓ 1.22
→ -0.07	→ 0.35	→ 0.51	→ 0.99	→ 1.37	→ 2.01	↓ 2.6	↓ 3.31	↓ 2.66	← 1.97
→ 0.09	→ 0.32	→ 0.77	→ 1.09	→ 1.83	→ 2.48	→ 3.35	✛ 4.35	← 3.46	← 2.62
→ -0.1	↑ 0.23	↑ 0.29	→ 0.57	→ 1.42	→ 2.31	→ 2.89	↓ 3.77	↓ 4.07	↓ 3.53
→ -0.17	↑ -0.04	↑ -0.21	→ -4.57	→ 1.4	→ 2.4	↓ 3.66	↓ 4.53	↓ 5.49	↓ 4.68
→ -0.24	→ 0.03	→ 0.07	→ 1.01	→ 2.1	→ 3.58	→ 4.52	↓ 5.97	↓ 7.04	↓ 6.07
→ -0.25	↑ -0.14	→ 0.38	→ 0.73	→ 3.21	→ 4.39	→ 5.95	→ 7.35	↓ 9	↓ 7.52
→ -0.32	↑ -0.21	↑ -0.99	→ -8.31	→ 2.66	→ 5.32	→ 6.97	→ 8.98	✛ 11.35	← 9.1
→ -0.33	↑ -0.22	→ 0.35	→ 0.75	→ 3.16	→ 4.41	→ 5.94	→ 7.37	↑ 9	↑ 7.53
↑ -0.53	→ -0.17	→ 0.23	→ 1.24	→ 2.19	→ 3.47	→ 4.6	↑ 5.97	↑ 7.13	↑ 6.12

$V_{10}(x,y)$ (Iteration 10)

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World

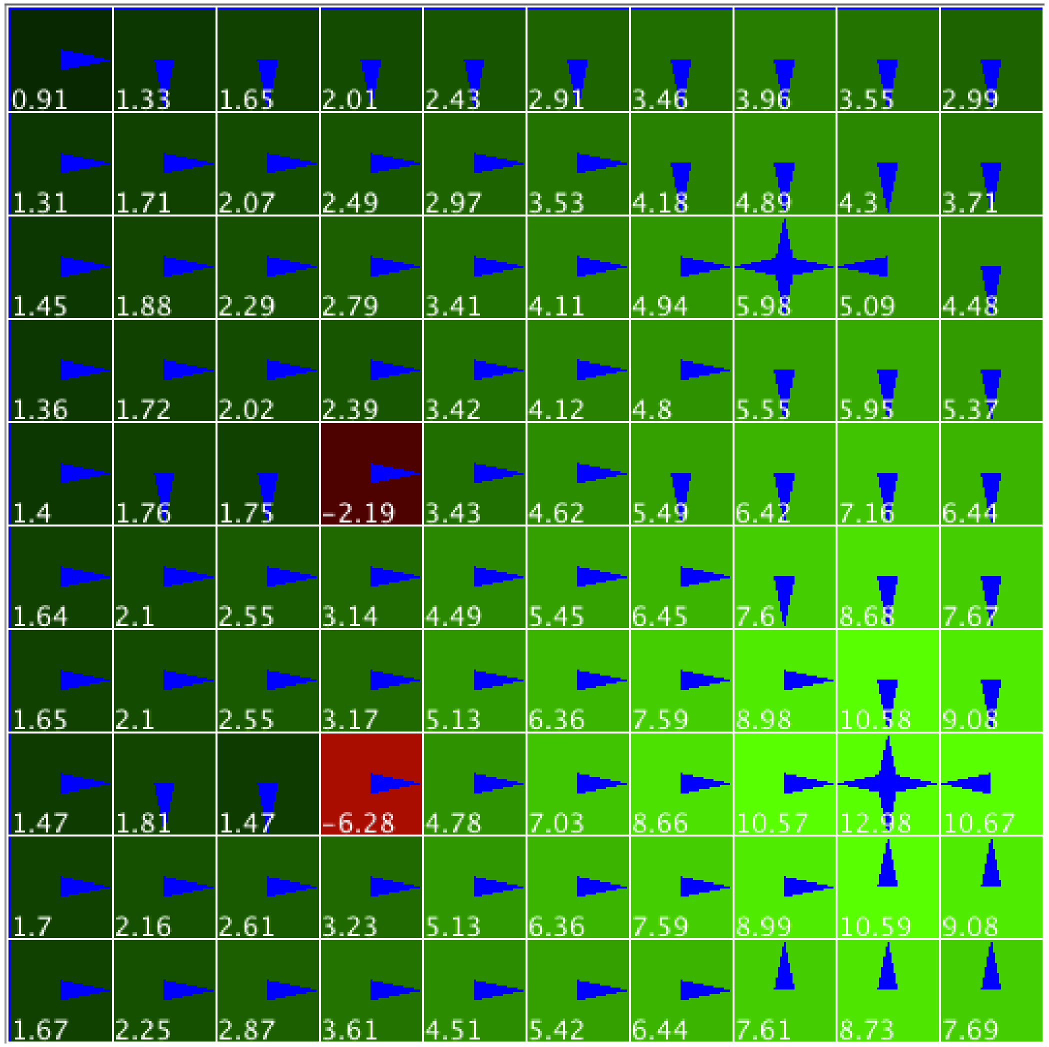


→ 0.37	↓ 0.76	↓ 1.09	↓ 1.43	↓ 1.85	↓ 2.32	↓ 2.86	↓ 3.37	↓ 2.97	↓ 2.42
→ 0.74	→ 1.15	→ 1.49	→ 1.91	→ 2.38	→ 2.93	↓ 3.58	↓ 4.29	↓ 3.71	↓ 3.14
→ 0.88	→ 1.29	→ 1.72	→ 2.2	→ 2.82	→ 3.51	→ 4.35	✛ 5.39	← 4.5	↓ 3.9
→ 0.74	→ 1.14	→ 1.42	→ 1.82	→ 2.83	→ 3.54	→ 4.21	↓ 4.96	↓ 5.36	↓ 4.78
→ 0.79	↓ 1.12	↓ 1.17	→ -2.78	→ 2.85	→ 4.04	↓ 4.9	↓ 5.82	↓ 6.56	↓ 5.84
→ 1	→ 1.52	→ 1.95	→ 2.57	→ 3.9	→ 4.87	→ 5.85	↓ 7	↓ 8.08	↓ 7.07
→ 1.04	→ 1.48	→ 1.97	→ 2.58	→ 4.54	→ 5.77	→ 6.99	→ 8.38	↓ 9.99	↓ 8.48
→ 0.82	↓ 1.21	↓ 0.87	→ -6.87	→ 4.19	→ 6.43	→ 8.06	→ 9.98	✛ 12.39	← 10.07
→ 1.08	→ 1.55	→ 2.03	→ 2.64	→ 4.54	→ 5.77	→ 6.99	→ 8.39	↑ 9.99	↑ 8.48
→ 1.05	→ 1.66	→ 2.29	→ 3.03	→ 3.92	→ 4.83	→ 5.84	↑ 7.01	↑ 8.13	↑ 7.08

$V_{20}(x,y)$ (Iteration 20)

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Value iteration example: Grid World



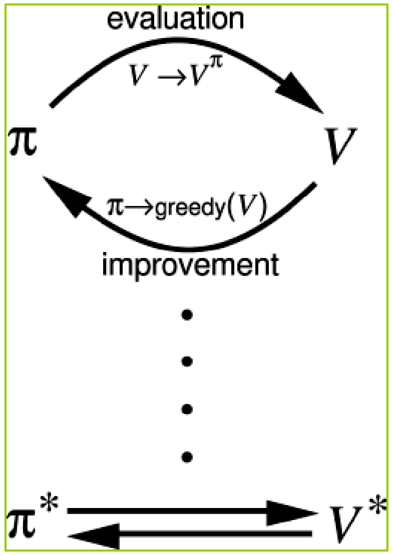
→ 0.91	↓ 1.33	↓ 1.65	↓ 2.01	↓ 2.43	↓ 2.91	↓ 3.46	↓ 3.96	↓ 3.55	↓ 2.99
→ 1.31	→ 1.71	→ 2.07	→ 2.49	→ 2.97	→ 3.53	↓ 4.18	↓ 4.89	↓ 4.3	↓ 3.71
→ 1.45	→ 1.88	→ 2.29	→ 2.79	→ 3.41	→ 4.11	→ 4.94	✛ 5.98	← 5.09	↓ 4.48
→ 1.36	→ 1.72	→ 2.02	→ 2.39	→ 3.42	→ 4.12	→ 4.8	↓ 5.55	↓ 5.95	↓ 5.37
→ 1.4	↓ 1.76	↓ 1.75	→ -2.19	→ 3.43	→ 4.62	↓ 5.49	↓ 6.42	↓ 7.16	↓ 6.44
→ 1.64	→ 2.1	→ 2.55	→ 3.14	→ 4.49	→ 5.45	→ 6.45	↓ 7.6	↓ 8.68	↓ 7.67
→ 1.65	→ 2.1	→ 2.55	→ 3.17	→ 5.13	→ 6.36	→ 7.59	→ 8.98	↓ 10.58	↓ 9.08
→ 1.47	↓ 1.81	↓ 1.47	→ -6.28	→ 4.78	→ 7.03	→ 8.66	→ 10.57	✛ 12.98	← 10.67
→ 1.7	→ 2.16	→ 2.61	→ 3.23	→ 5.13	→ 6.36	→ 7.59	→ 8.99	↑ 10.59	↑ 9.08
→ 1.67	→ 2.25	→ 2.87	→ 3.61	→ 4.51	→ 5.42	→ 6.44	↑ 7.61	↑ 8.73	↑ 7.69

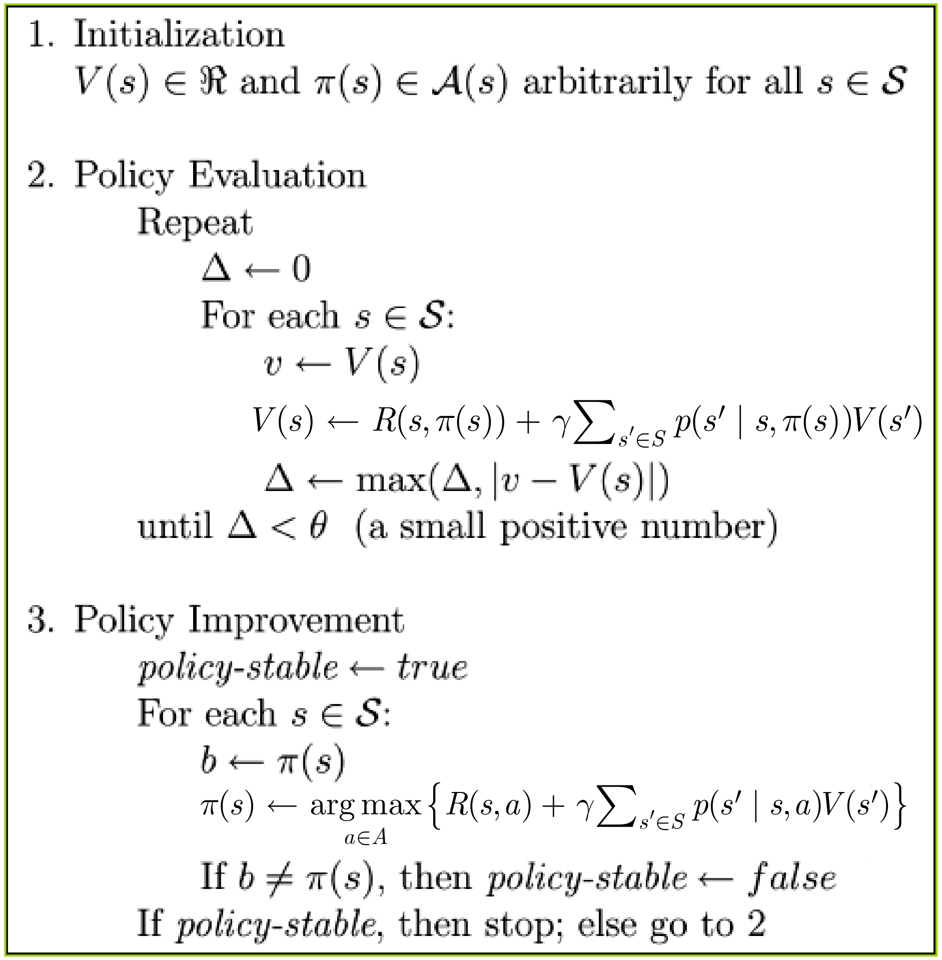
$V_{50}(x,y)$ (Iteration 50)

10 x 10 Grid World Java Applet: http://www.cs.ubc.ca/~poole/demos/mdp/vi.html

Policy iteration

- Iteratively evaluate V^{π} and improve the policy π
- A finite MDP has a finite number of policies, so convergence is guaranteed in a finite number of iterations





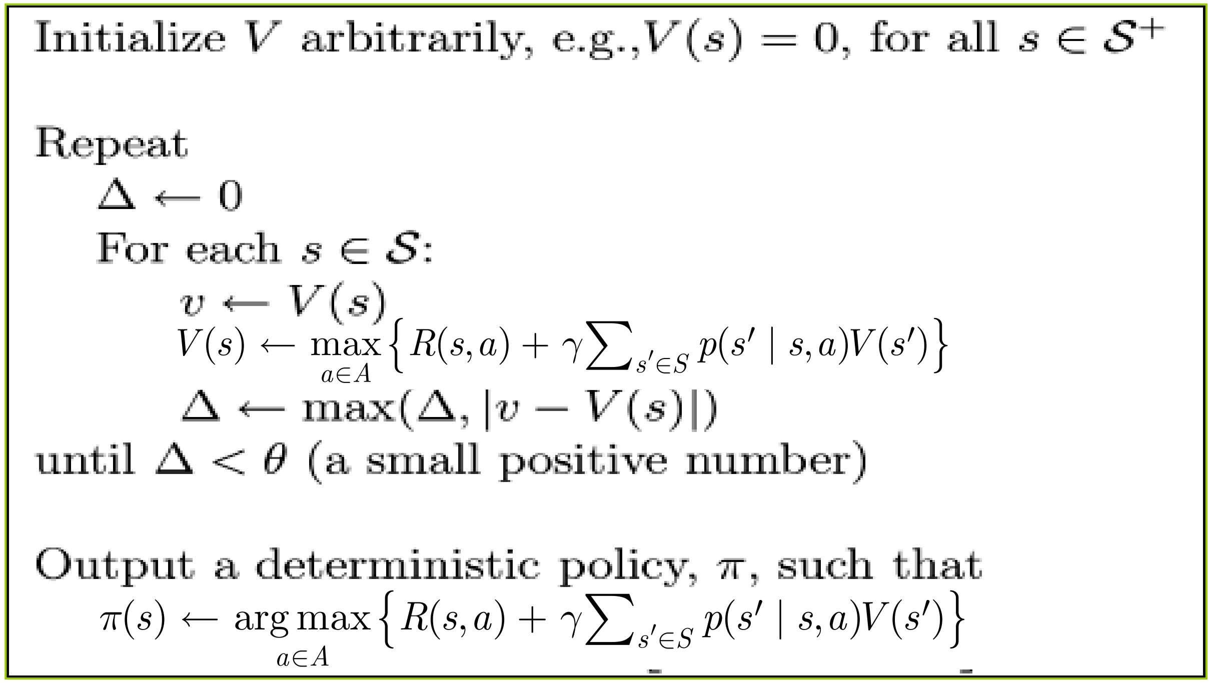
```
1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
       Repeat
           $\Delta \leftarrow 0$
           For each $s \in \mathcal{S}$:
               $v \leftarrow V(s)$
               $V(s) \leftarrow R(s,\pi(s)) + \gamma \sum_{s' \in S} p(s' \mid s, \pi(s)) V(s')$
               $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
       until $\Delta < \theta$  (a small positive number)

3. Policy Improvement
       policy-stable $\leftarrow$ true
       For each $s \in \mathcal{S}$:
           $b \leftarrow \pi(s)$
           $\pi(s) \leftarrow \arg\max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s, a) V(s') \right\}$
           If $b \neq \pi(s)$, then policy-stable $\leftarrow$ false
       If policy-stable, then stop; else go to 2
```

Policy iteration discussion

- There is a finite number of stationary policies:
  - A total of $|S|$ states and $|A|$ actions possible in each state → A total of $|A|^{|S|}$ unique stationary policies
- Policy iteration converges in a finite number of steps
- Policy evaluation can be done iteratively (as shown before) or by solving a system of linear equations
- When the state space is large, this is expensive

Relationship between policy iteration and value iteration

Recall value iteration...



```
Initialize $V$ arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right\}$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
    $\pi(s) \leftarrow \arg\max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V(s') \right\}$
```

Value iteration truncates policy iteration by combining one sweep of policy evaluation and one of policy improvement in each of its sweeps.

Relationship between policy iteration and value iteration

- Each iteration of value iteration is relatively cheap compared to iterations of policy iteration because policy iteration requires solving a system of $|S|$ linear equations in each iteration.
- However, the trade off is that policy iteration requires less iterations to converge.
- Implementations in MATLAB
  - **Value iteration:** “slow” because MATLAB is bad at iterative computations
  - **Policy iteration:** “fast” because MATLAB is very good at solving systems of linear equations

Complete knowledge

- In problems of *complete knowledge*, the agent has a complete and accurate model of the environment's dynamics.
- If the environment is an MDP, then such a model consists of the one-step *transition probabilities* and *expected rewards* for all states and their allowable actions.
- In problems of *incomplete knowledge*, a complete and perfect model of the environment is not available.

What happens if the environment is unknown?

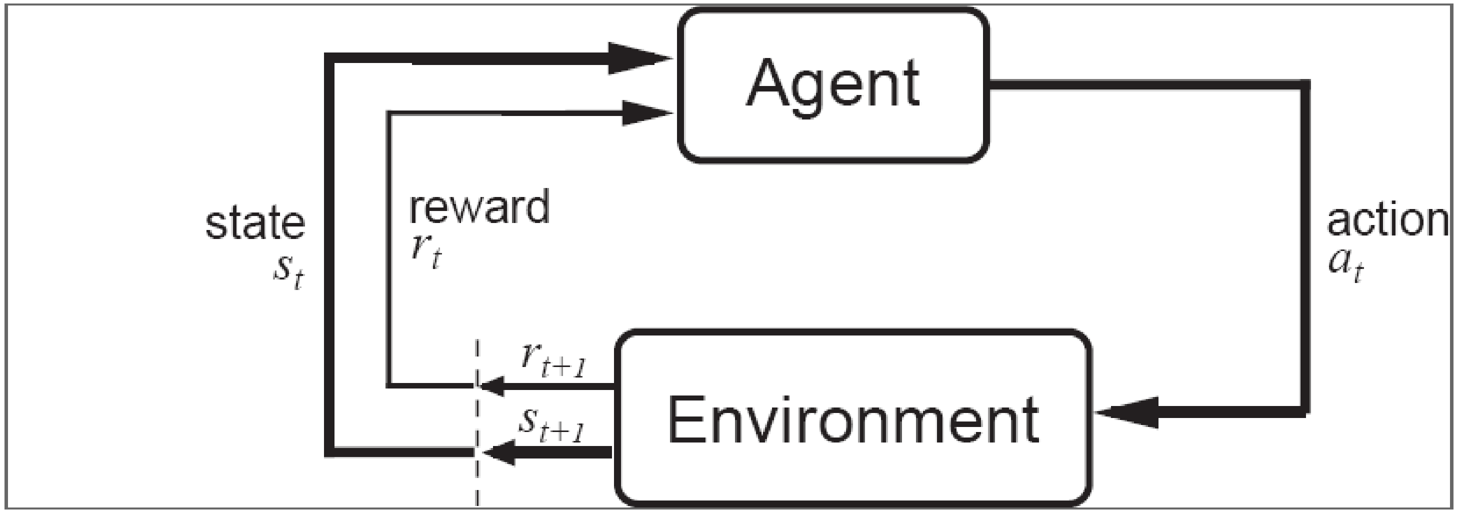
- Model-based
  - Learn reward and transition probabilities
  - Then compute optimal value function and corresponding policy
  - Example: RTDP (real-time dynamic programming)
- Model-free
  - Learn value function or action value function directly

Why is Online Learning Important?

- If the state transition probabilities are known, finding the optimal policy becomes a straightforward computational problem, however…
- If the transition probabilities and/or rewards are unknown, then this is a *reinforcement learning* (RL) problem.
- Reinforcement-learning techniques learn from experience, no knowledge of the model is required.

Agent-environment interaction

By "state" we mean whatever information is available to the agent

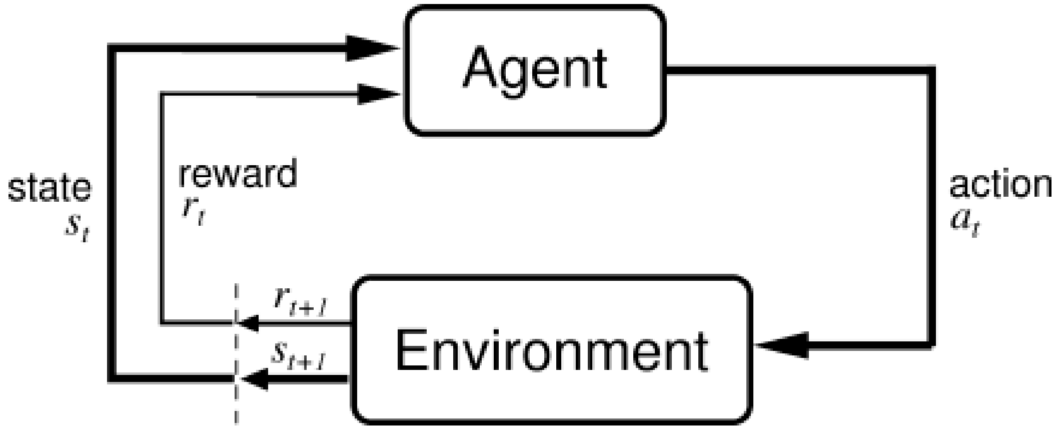


Everything inside the agent is completely known and controllable by the agent; everything outside is incompletely controllable but may or may not be completely known.

In our setting, who are the agent and the environment?
What are the states, actions, rewards?

Typical Agent

- In reinforcement learning (RL), the agent observes a state and takes an action.
- Afterward, the agent receives a reward.



Goal: Optimize Cumulative Reward

- Rewards are calculated in the environment
- Used to teach the agent how to reach a goal state
- Must signal what we ultimately want achieved, not necessarily subgoals
- May be *discounted* over time
- In general, seek to maximize the *expected return*

Examples of reinforcement learning techniques: Q-learning, Sarsa, actor critic etc.

Estimation of Action Values

- State values $V^{\pi}(s)$ are not enough without a model
- We need action values as well:

  $Q^{\pi}(s,a)$ → expected return when starting in state a, taking action a, and thereafter following policy π
- Exploration vs. Exploitation
  - The agent believes that a particular action has a high value; should it choose that action all the time, or should it choose another one that it has less information about, but seems to be worse?

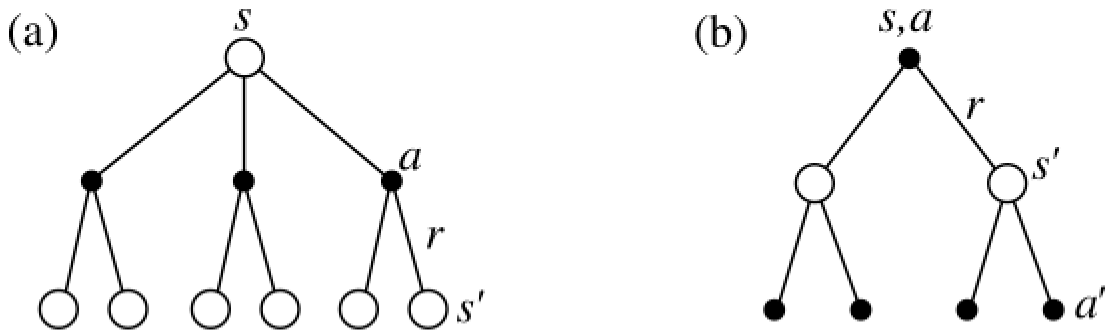
Another Value Function: State-action-value Function

- Q^π (also called Q-function) is the expected return (value) starting from state *s*, taking action *a*, and thereafter following policy π

State-action-value function for policy π

$$Q^\pi(s,a) = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t R_t \mid s_0 = s, a_0 = a\right]$$
$$= R(s,a) + \gamma\sum_{s'\in\mathcal{S}} p\left(s' \mid s,a\right)V^\pi(s')$$

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



Backup diagrams for (a) V^π and (b) Q^π

Q-learning

- Q-learning algorithm does not specify what the agent should actually do.
- The agent learns a *Q*-function that can be used to determine an optimal action.
- There are two things that are useful for the agent to do:
  - **exploit** the knowledge that it has found for the current state *s* by chosing one of the actions *a* that maximizes *Q[s,a]*.
  - **explore** in order to build a better estimate of the optimal *Q*-function. That is, it should select a different action from the one that it currently thinks is best.

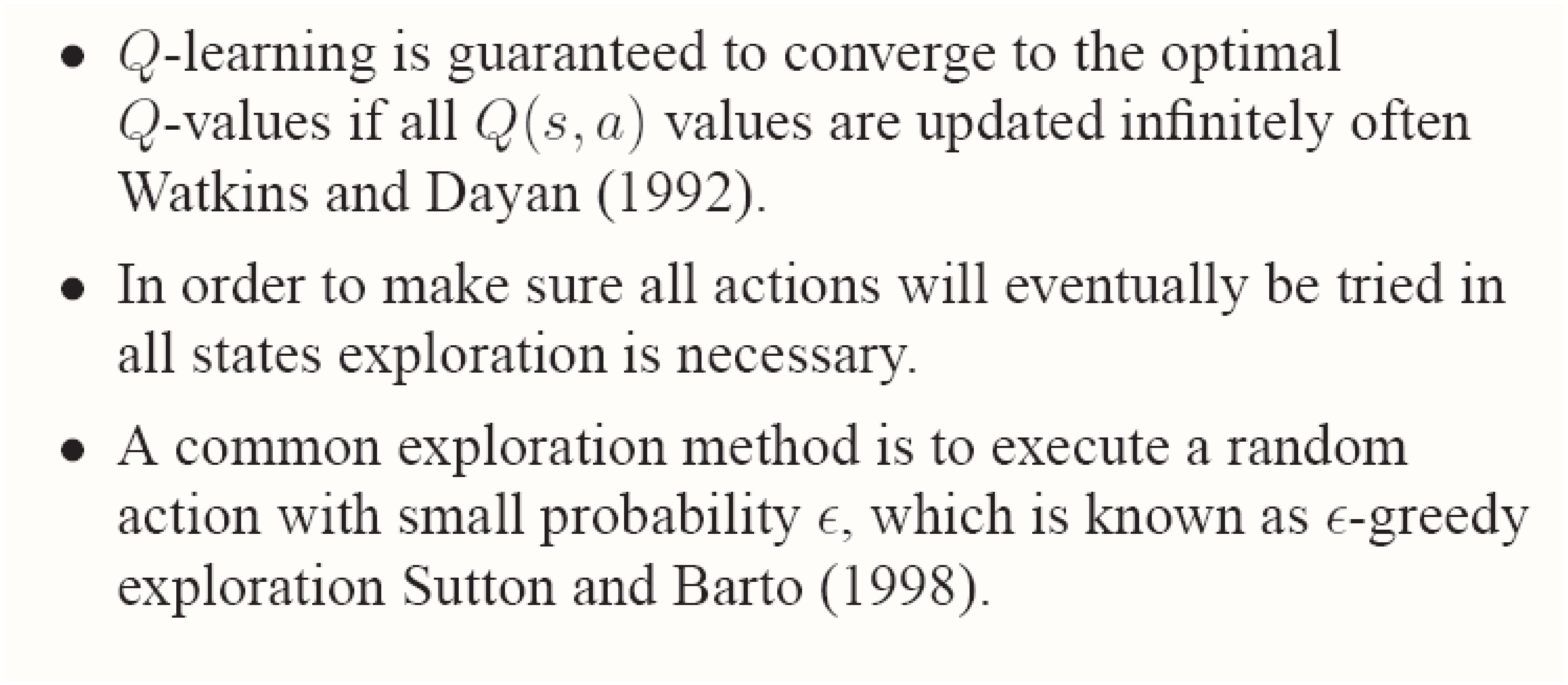
Q-learning

- At each step (denote by s the current state), choose the action a which maximizes the function Q(s, a)
- Q is the estimated utility function – it tells us how good an action is given a certain state (include the estimated future utility)
- Q(s, a) = immediate reward for making an action + best utility (Q) for the resulting state
- Q learning:

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

where r_{t+1} is the reward observed after performing a_t in s_t, and where α_t is the learning rate ($0 < \alpha_t \leq 1$).

Q-learning discussion



- Q-learning is guaranteed to converge to the optimal Q-values if all $Q(s, a)$ values are updated infinitely often Watkins and Dayan (1992).
- In order to make sure all actions will eventually be tried in all states exploration is necessary.
- A common exploration method is to execute a random action with small probability ϵ, which is known as ϵ-greedy exploration Sutton and Barto (1998).

Other reinforcement learning techniques

- Sarsa, Actor critic, etc.
- However, this goes beyond the topic of this class

Problems with Generic Learning Algorithms

- Curse of Dimensionality (often multi-dimensional state space)
- Slow convergence (not useful for real-time applications)
- Solutions to these problems exist