

南京大學

工程管理学院



智能控制与机器人学

第一章 强化学习 (Reinforcement Learning)

南京大学控制与系统工程系

陈春林

Email: clchen@nju.edu.cn

2018年10月29日

参考书:

- Sutton R.S., Barto A.G. **Reinforcement Learning: An Introduction**. MIT Press, Cambridge, MA, 1998
-

第一章 强化学习

Chapter 1. Reinforcement Learning

Contents

1 Introduction

2 Elementary Solution Methods

2.1 Dynamic Programming

2.2 Monte Carlo Methods

2.3 Temporal-Difference Learning

3 A Unified View

3.1 Eligibility Traces

3.2 Generalization & Function Approximation

3.3 Planning & Learning

1. Introduction

- RL
 - Evaluative Feedback
 - RL Problem
-

What is Reinforcement Learning?

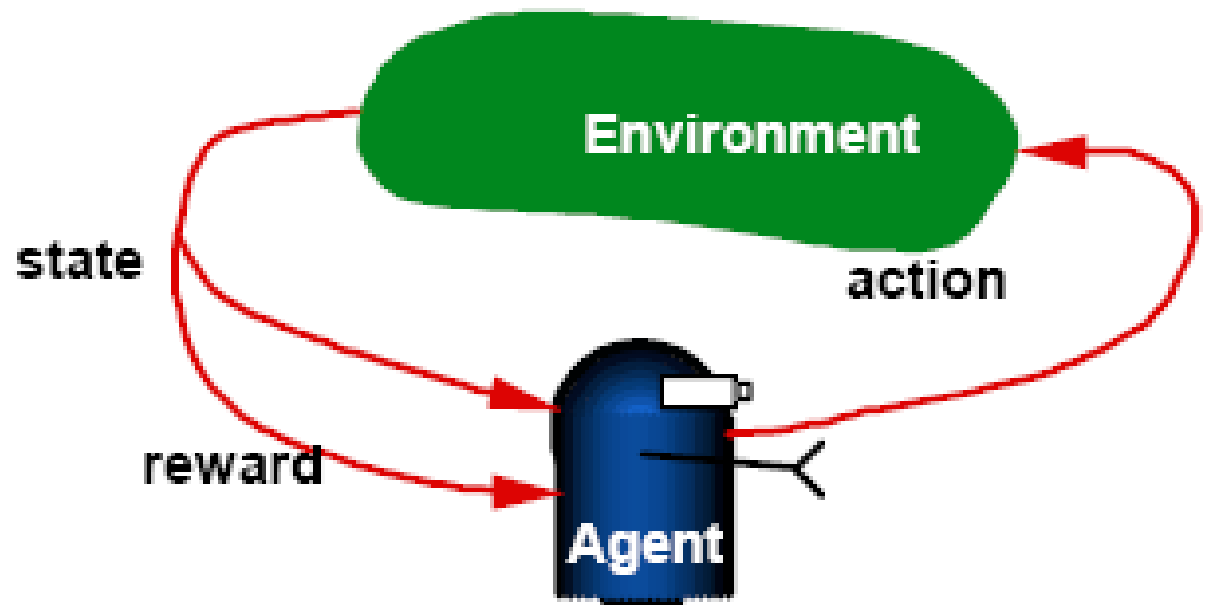
- A conceptually simple unifying framework for artificial intelligence
 - RL is much more than a model of “learning”
 - It also addresses planning under uncertainty, sequential decision-making, real-time search, acting to gain info...
 - It provides a formal framework for cognitive science
 - Animal and human individual and social behavior
 - Neuroscience – reward learning in the brain
 - Powerful simulation-based control/optimization framework
 - Many state-of-the-art applications
-

What is Reinforcement Learning?

- Learning what to do—how to map situations to actions—so as to maximize a *scalar* reward signal
 - Kierkegaard: Life can only be understood by going *backwards*, but must be lived going *forwards*.
 - Key insight due to Arthur Samuel [1959]:
 - Adjust the evaluation of states *early* in the game to match those encountered *later* during *actual* lines of play
 - Modern reinforcement learning research is based on a precise formalization of this principle
-

RL Agent Design

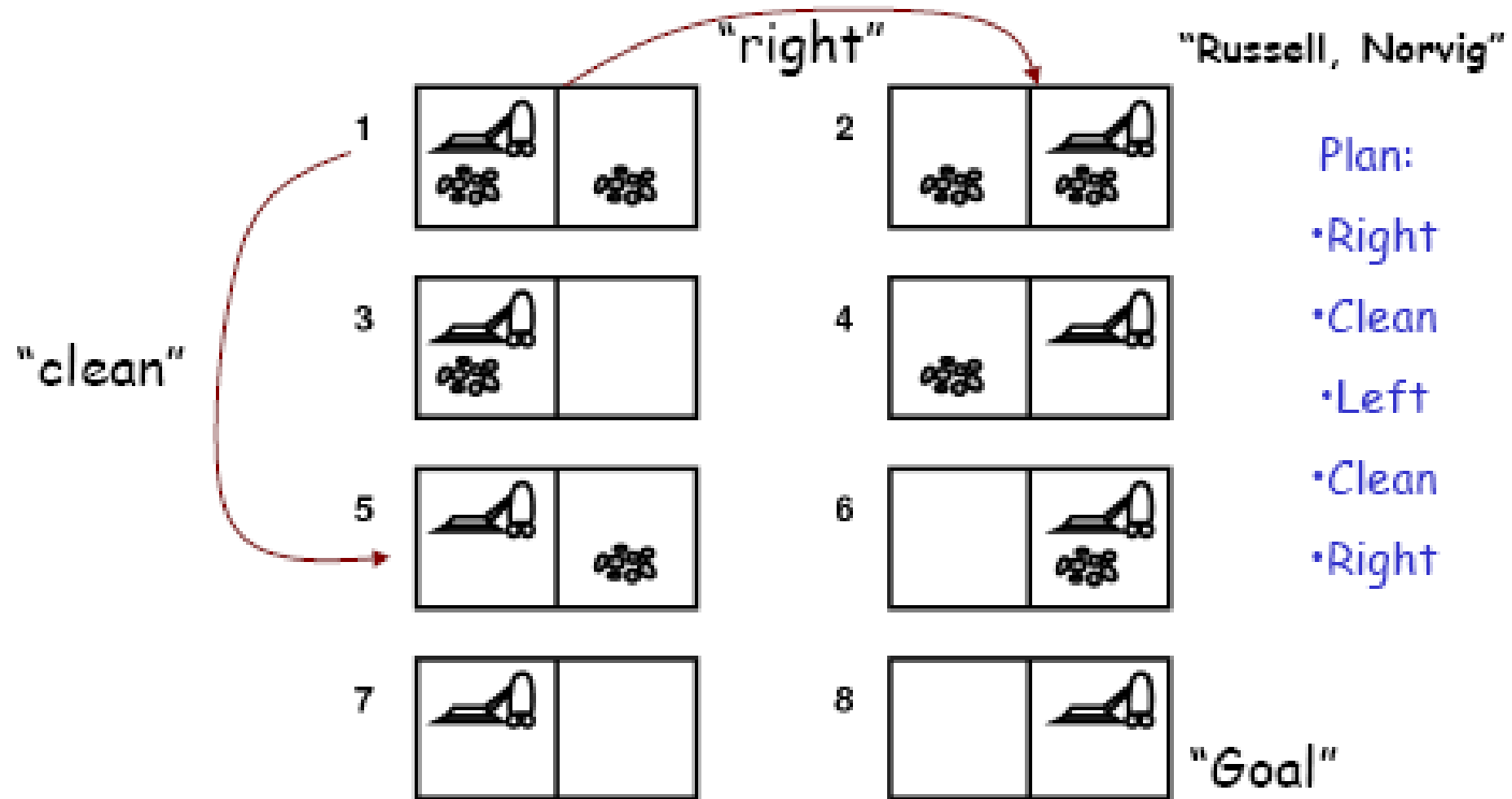
- ❑ Environment is stochastic: actions have uncertain outcomes
- ❑ Sense-act loop:
 - Perceive state of environment
 - Choose action using behavior π
 - Receive reward
 - Adjust π



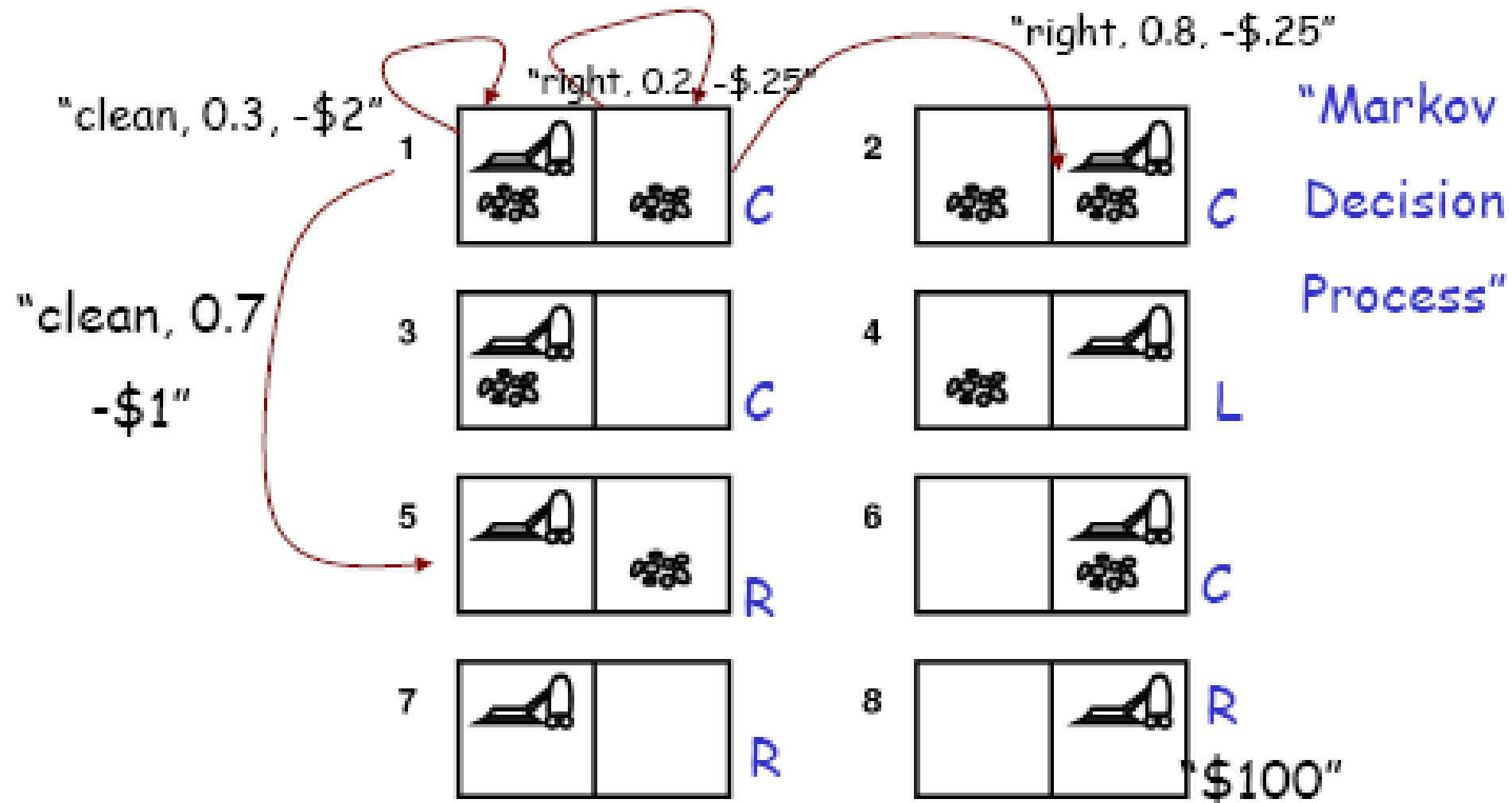
Connections between AI/ML and RL

- Search
 - Arc costs
 - A* search ($f = g + h$)
 - Real-time A*
 - Learning Real-time A*
 - Planning
 - Goals
 - Action models (STRIPS)
 - Conditional planning
 - Supervised Learning
 - Learning from instruction
 - Structural credit assignment
 - Dynamic programming
 - Expected one-step reward
 - Value iteration (Bellman)
 - Real-time DP
 - Q-learning
 - Model-based RL
 - Reward functions
 - Transition models
 - Closed-loop policies
 - Temporal-difference Learning
 - Learning from evaluation
 - Temporal credit assignment
 - Exploration-exploitation tradeoff
-

State-Action Model



Stochastic actions & Random Payoffs



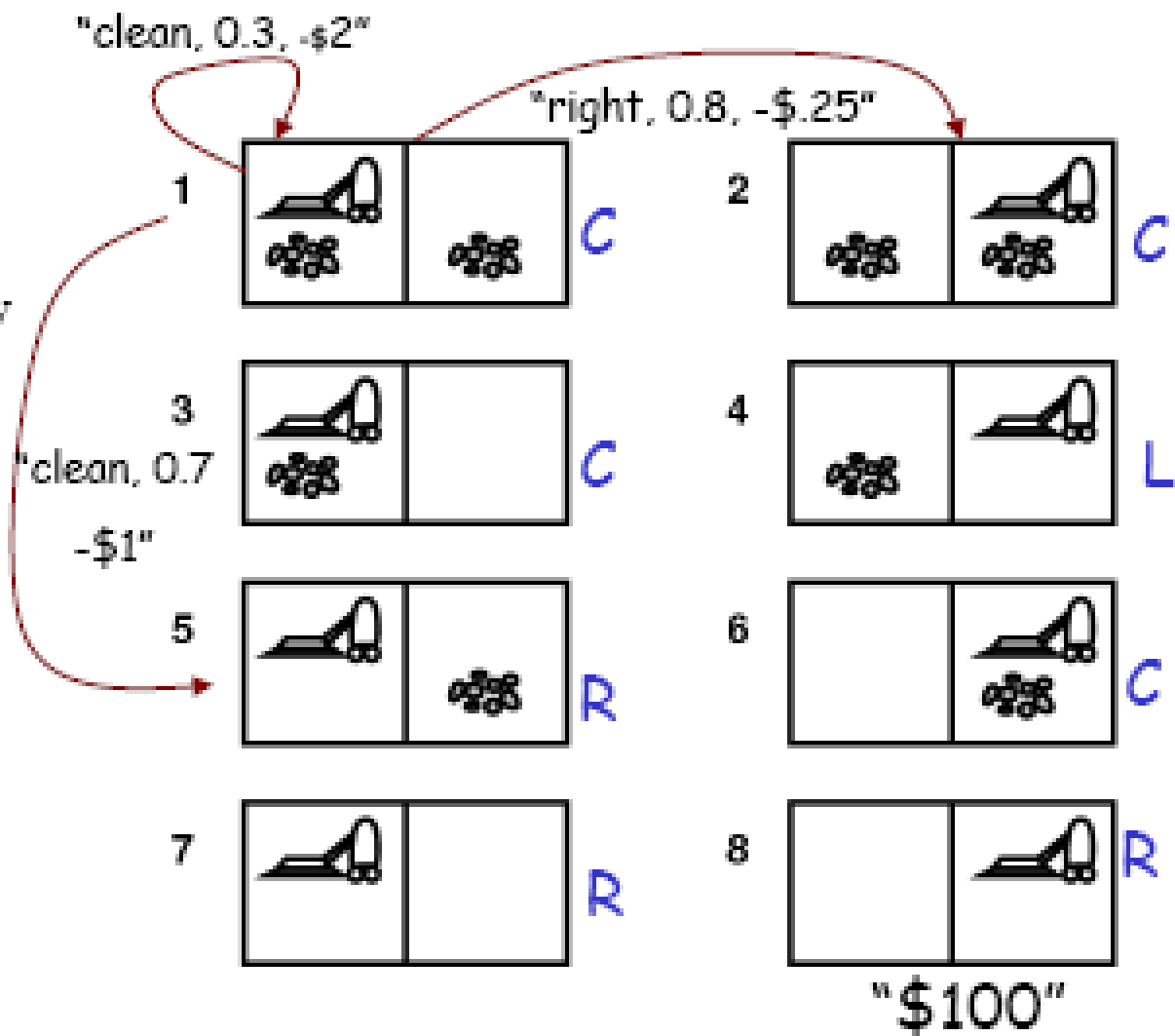
Maximize Expected Utility

Expected reward $r(1, \text{"clean"})$

$$= 0.3 * -\$2 + 0.7 * -\$1 = -\$1.3$$

The agent should find a policy that maximizes the expected sum of rewards

Maximize $E \left[\sum_t \gamma^t r_t \right]$

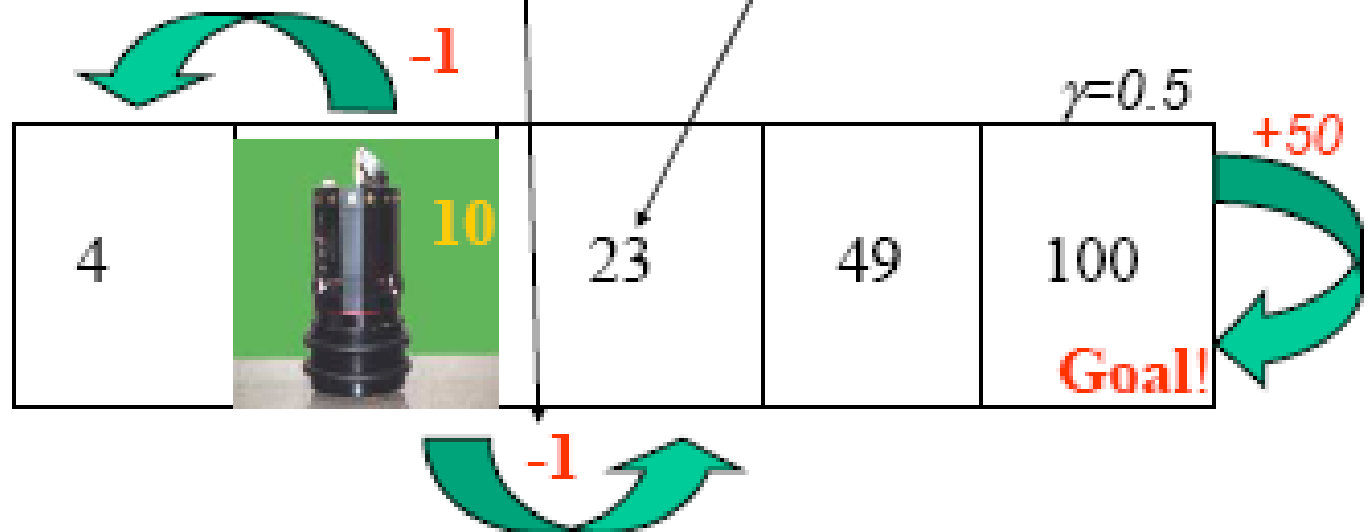


The Bellman Optimality Equation

- A **policy** is a mapping from states to actions
- **Optimal** policies maximize the long-term discounted reward sum, and define the same **value function** over states

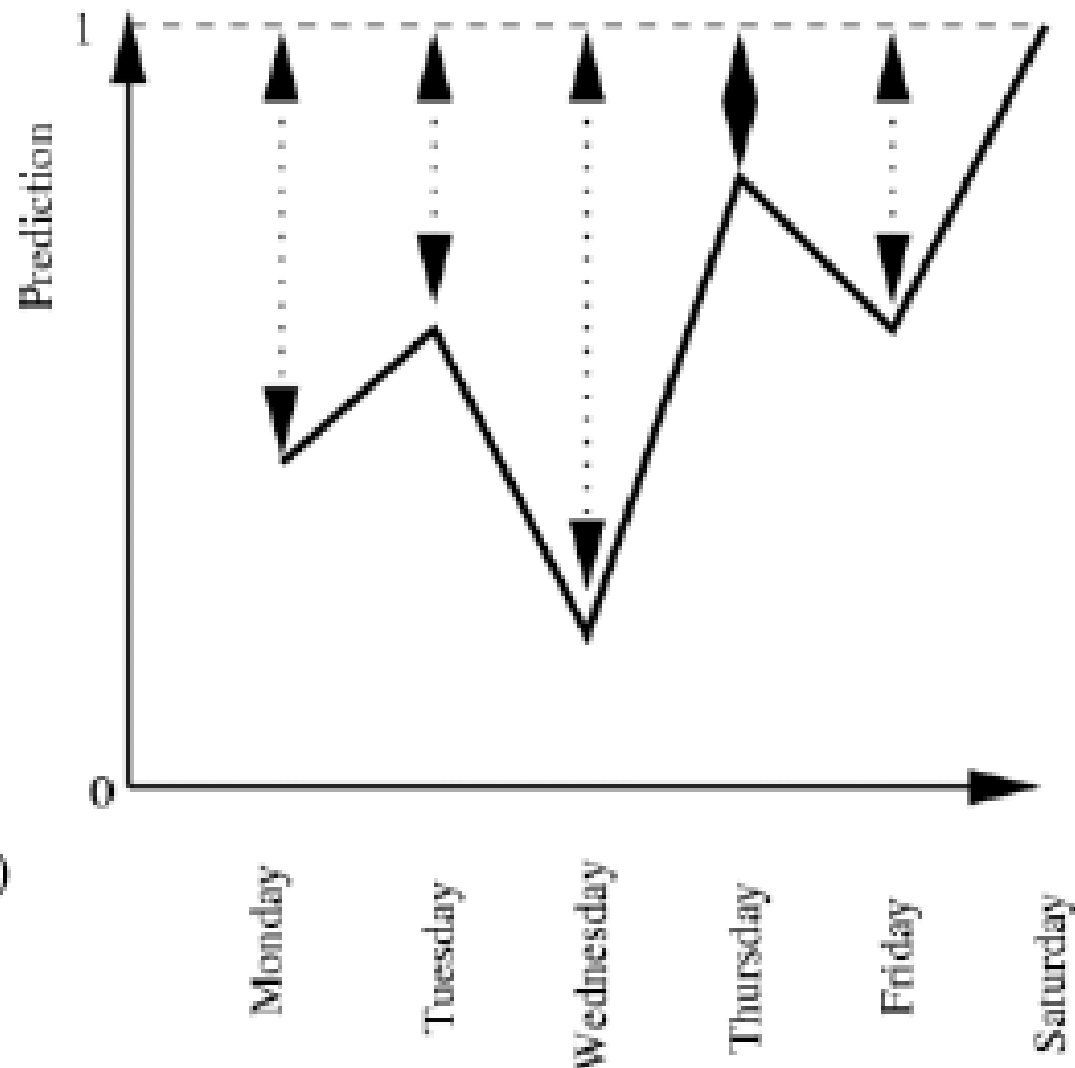
$$V^*(s) = \max_{a \in A(s)} \left(\sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V^*(s')) \right)$$

Choose
greedy action
given V^*



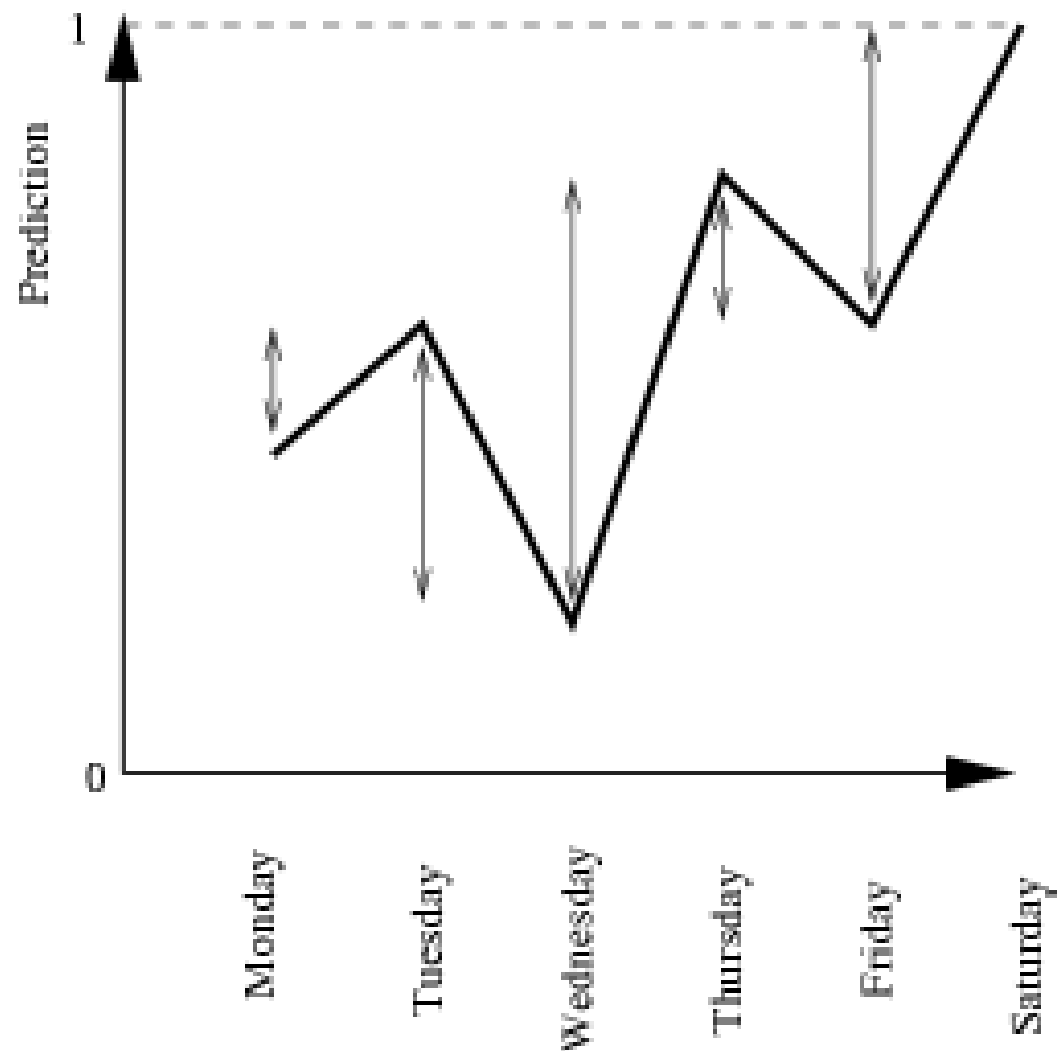
Weather Prediction: Supervised Learning

- ❑ Consider the problem of predicting Saturday's weather
- ❑ Each day, we are given measurements of humidity, temperature etc.
- ❑ The prediction is some function (unknown) of the data for that day
- ❑ Learning is based on comparing the predictions made on each day with the final outcome
- ❑ Example: Delta rule (neural nets)
- ❑ Non-incremental



Weather Prediction using TD Learning

- TD learning is based on comparing today's prediction with tomorrow's prediction
- Incremental approach since learning can occur before final outcome is known
- Why does this work?

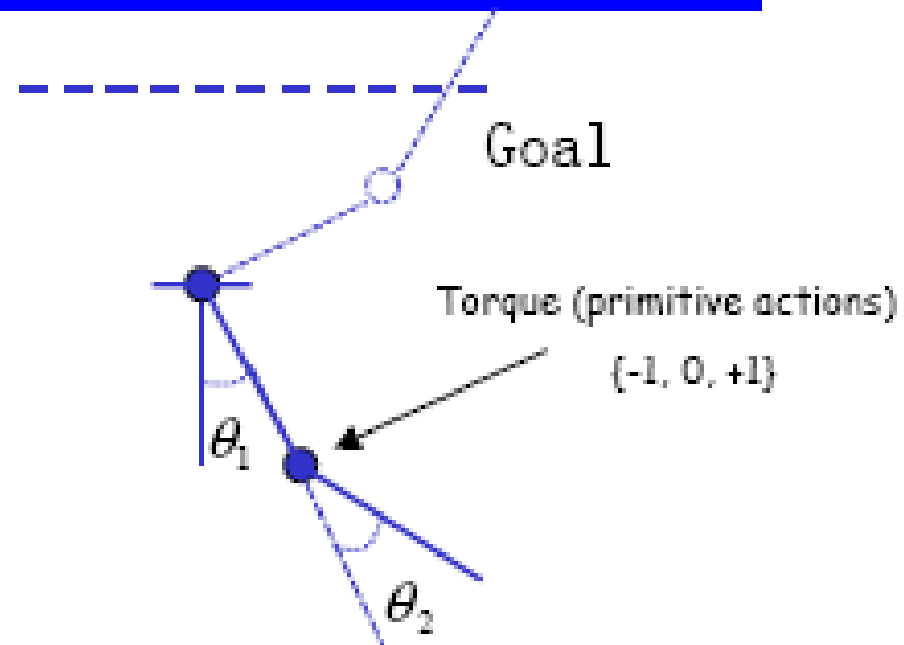


RL for Acrobat Task

(Sutton, 1996; Rohanimanesh, 2001)

Continuous
State vector

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \cdot \\ \dot{\theta}_1 \\ \cdot \\ \dot{\theta}_2 \\ \tau \end{bmatrix}$$



Prespecified Behaviors (for each +1,-1 torque, define a behavior)

- Can be initiated anywhere
- Policy for behavior(τ): Apply torque τ (+1, or -1)
- Terminate when sign of angular velocity of 2nd link disagrees with torque

Teaching Humanoid Robots

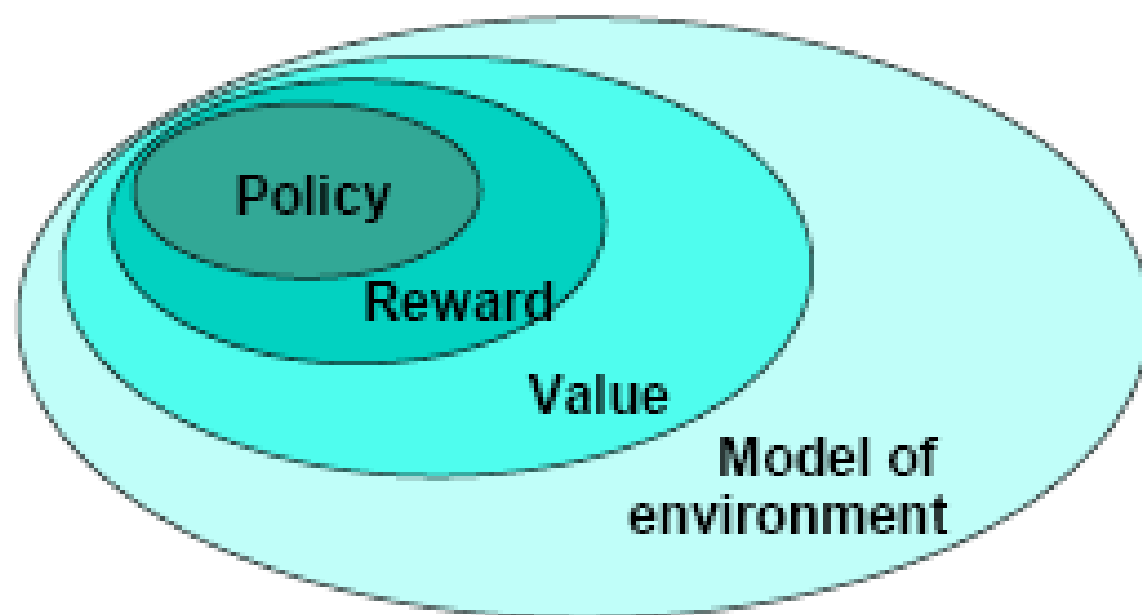
- ☐ Teaching Humanoids
 - High-DOF behaviors
 - Limited perception
 - Significant uncertainty
 - Delayed feedback
- ☐ Very difficult to program
- ☐ Supervised learning is of limited value



“Robonaut”

DARPA Robot-2020 Project: NASA, U.Mass, MIT, USC, Vanderbilt

Elements of RL

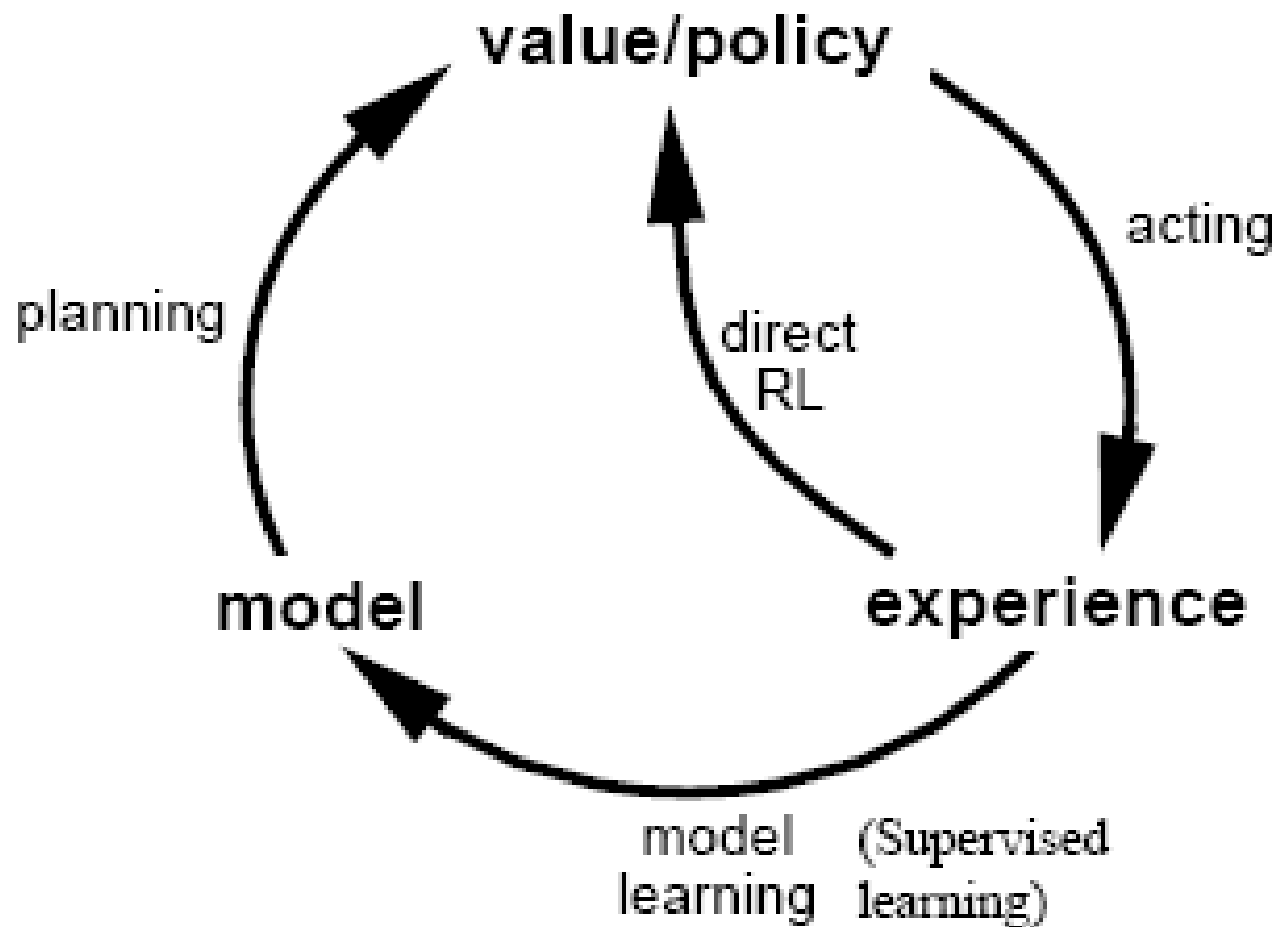


- ❑ **Policy:** what to do
 - ❑ **Reward:** what is good
 - ❑ **Value:** what is good because it *predicts* reward
 - ❑ **Model:** what follows what
-

Key Features of RL

- ❑ Agent is not told which actions to take (the desired output)
 - ❑ Trial-and-Error search
 - ❑ Possibility of delayed reward
 - Sacrifice short-term gains for greater long-term gains
 - ❑ The need to *explore* and *exploit*
 - ❑ Considers the whole problem of a goal-directed agent interacting with an uncertain environment
-

Architecture of an RL Agent



An Extended Example: Tic-Tac-Toe

X						

O	X					

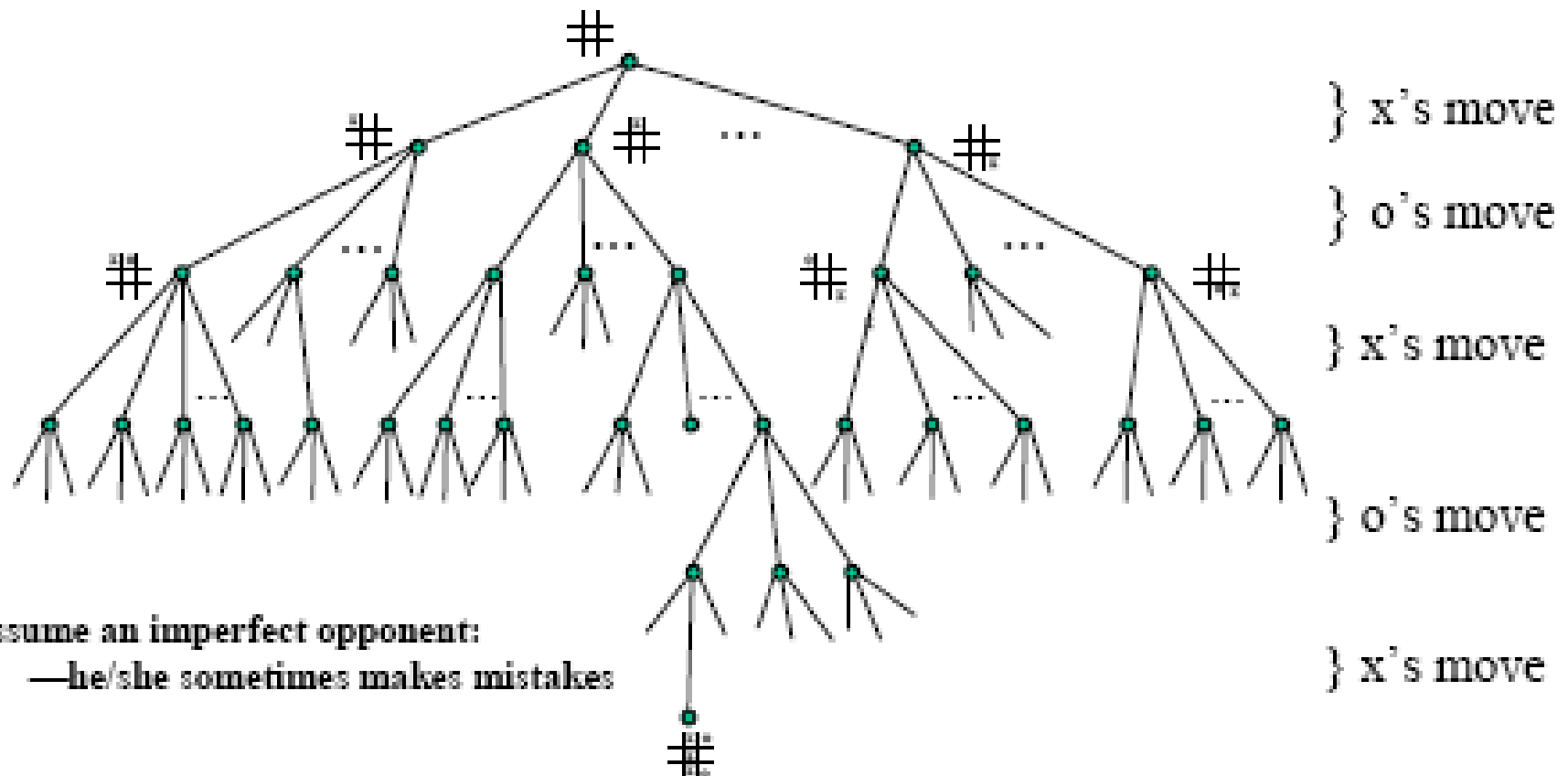
		X				
O	X					

		X				
O	X					
O						

X		X				
O	X					
O						






X	O	X				
O	X					
O						

X	O	X				
O	X					
O					X	

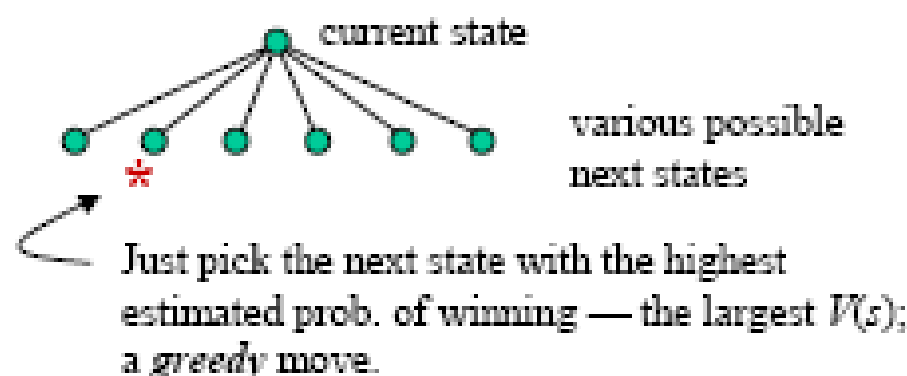


An RL Approach to Tic-Tac-Toe

1. Make a table with one entry per state:

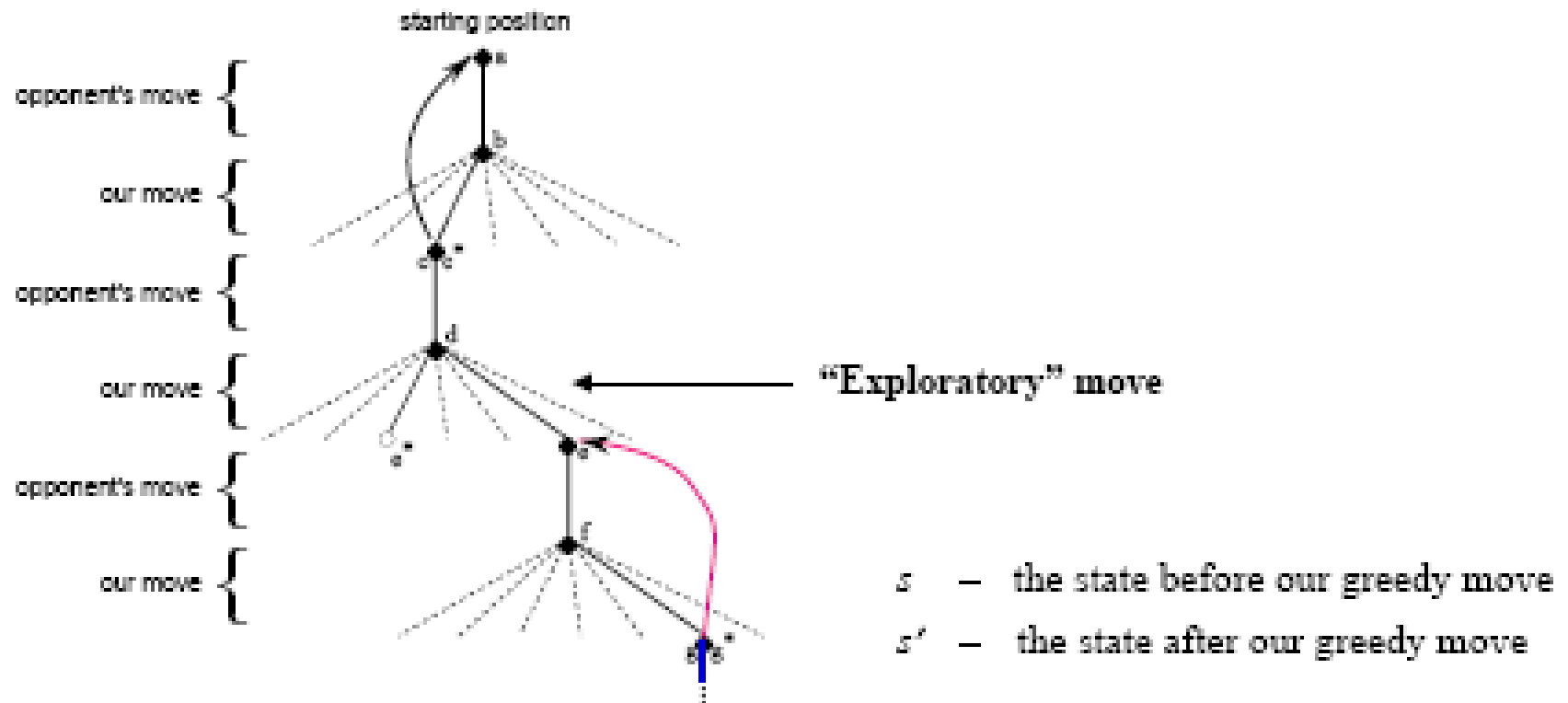
State	$V(s)$ – estimated probability of winning	
	.5	
	.5	
...	?	
	1	win
...	...	
	0	loss
...	...	
	0	draw

2. Now play lots of games.
To pick our moves,
look ahead one step:



But 10% of the time pick a move at random;
an *exploratory* move.

RL Learning Rule for Tic-Tac-Toe



We increment each $V(s)$ toward $V(s')$ – a *backup*:

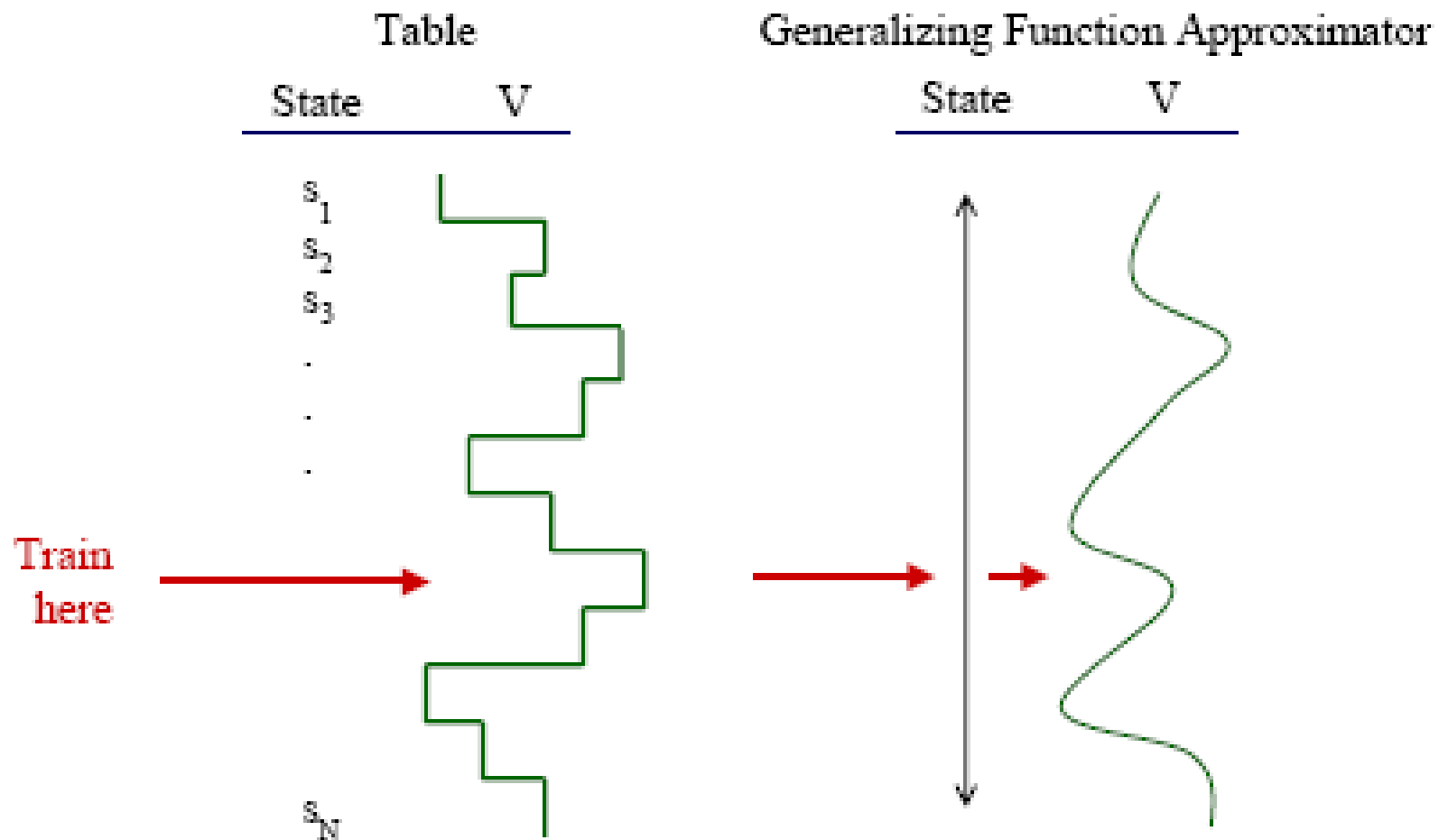
$$V(s) \leftarrow V(s) + \alpha[V(s') - V(s)]$$

↖ a small positive fraction, e.g., $\alpha = .1$
the *step-size parameter*

How can we improve this T.T.T. player?

- ☐ Take advantage of symmetries
 - representation/generalization
 - How might this backfire?
 - ☐ Do we need “random” moves? Why?
 - Do we always need a full 10%?
 - ☐ Can we learn from “random” moves?
 - ☐ Can we learn offline?
 - Pre-training from self play?
 - Using learned models of opponent?
 - ☐ ...
-

e.g. Generalization



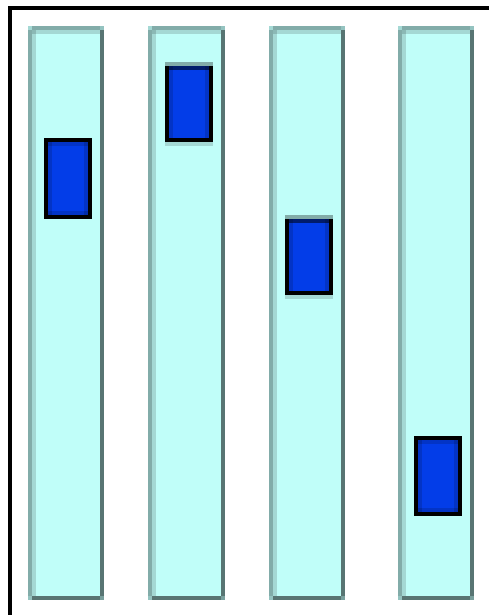
How is Tic-Tac-Toe Too Easy?

- ☐ Finite, small number of states
- ☐ One-step look-ahead is always possible
- ☐ State completely observable
- ☐ ...

Elevator Dispatching

Crites and Barto, 1996

10 floors, 4 elevator cars



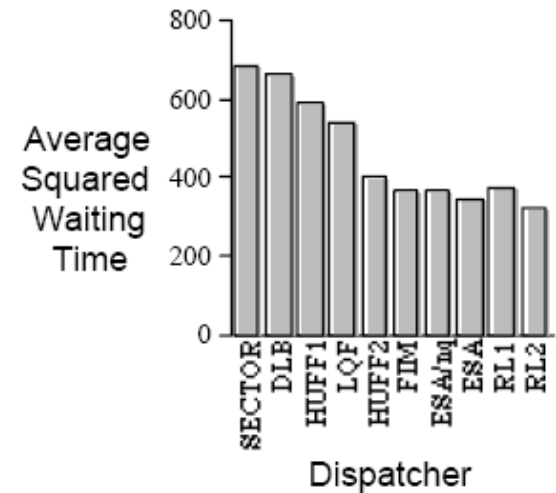
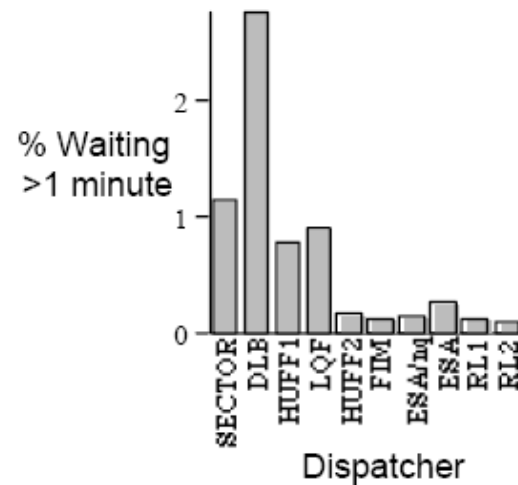
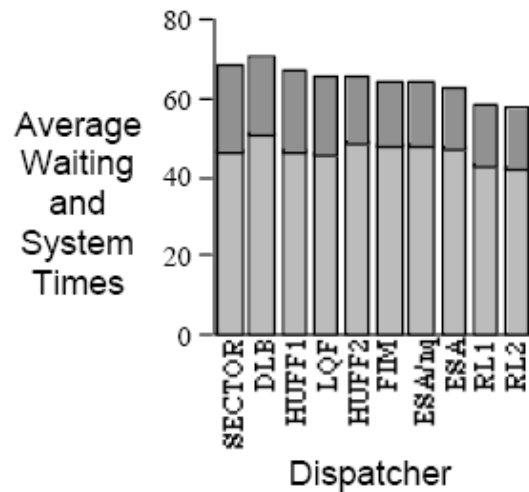
STATES: button states; positions, directions, and motion states of cars; passengers in cars & in halls

ACTIONS: stop at, or go by, next floor

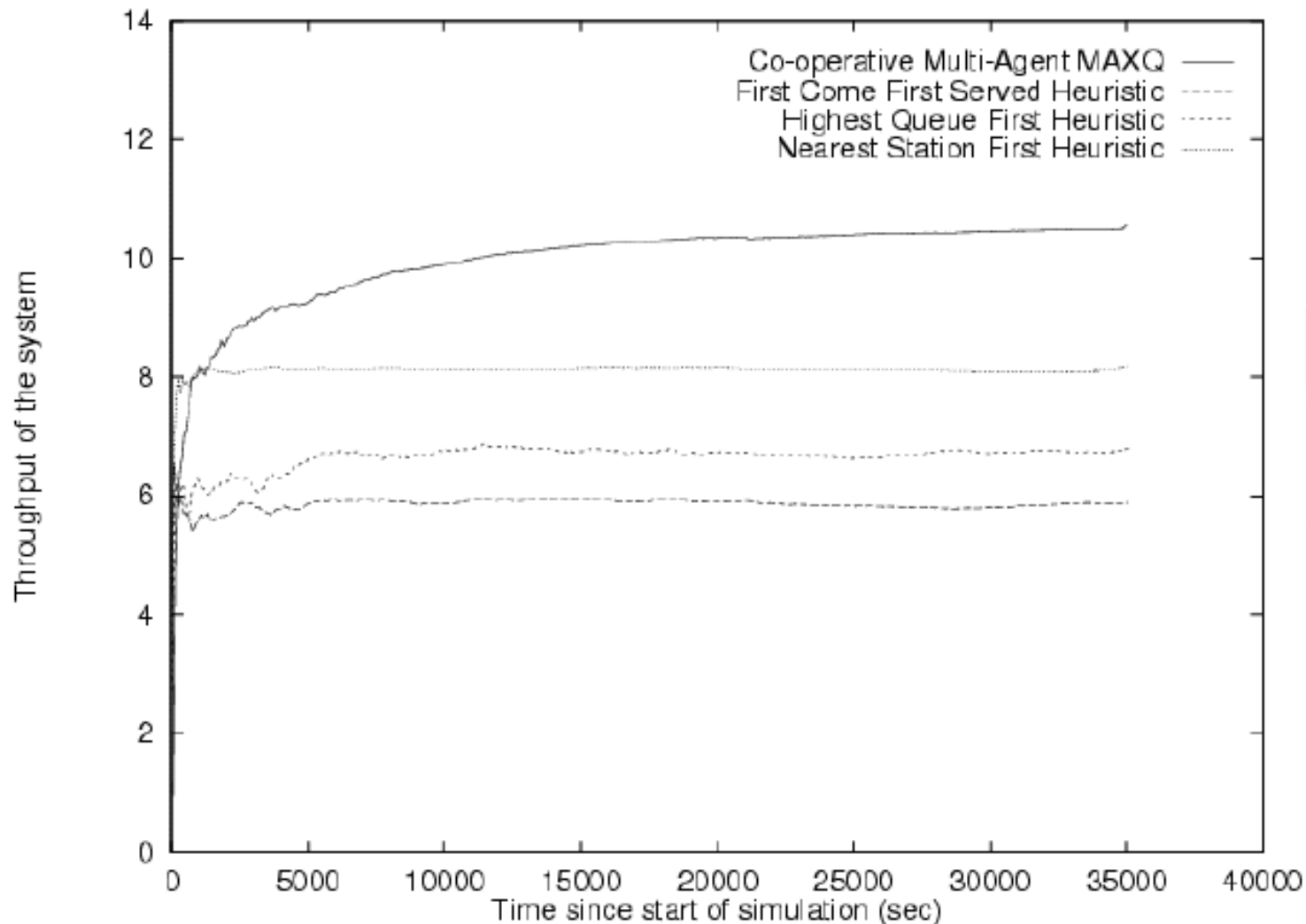
REWARDS: roughly, -1 per time step for each person waiting

Conservatively about 10^{22} states

Performance Comparison



Multi AGV Scheduling



Makar, et al,
Agents 2001



The Exploration/Exploitation Dilemma

- Suppose you form estimates

$$Q_t(a) \approx Q^*(a) \quad \text{action value estimates}$$

- The **greedy** action at t is

$$a_t^* = \operatorname{argmax}_a Q_t(a)$$

$$a_t = a_t^* \Rightarrow \text{exploitation}$$

$$a_t \neq a_t^* \Rightarrow \text{exploration}$$

- You can't exploit all the time; you can't explore all the time
 - You can never stop exploring; but you should always reduce exploring
-

Action-Value Methods

- Methods that adapt action-value estimates and nothing else, e.g.: suppose by the t -th play, action a had been chosen k_a times, producing rewards r_1, r_2, \dots, r_{k_a} , then

$$Q_t(a) = \frac{r_1 + r_2 + \dots + r_{k_a}}{k_a}$$

“sample average”

- $\lim_{k_a \rightarrow \infty} Q_t(a) = Q^*(a)$

ϵ -Greedy Action Selection

- Greedy action selection:

$$a_t = a_t^* = \arg \max_a Q_t(a)$$

- ϵ -Greedy:

$$a_t = \begin{cases} a_t^* & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

... the simplest way to try to balance exploration and exploitation

Softmax Action Selection

- Softmax action selection methods grade action probs. by estimated values.
- The most common softmax uses a Gibbs, or Boltzmann, distribution:

Choose action a on play t with probability

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^n e^{Q_t(b)/\tau}},$$

where τ is the

“computational temperature”

Incremental Implementation

Recall the sample average estimation method:

The average of the first k rewards is
(dropping the dependence on a):

$$Q_k = \frac{r_1 + r_2 + \cdots + r_k}{k}$$

Can we do this incrementally (without storing all the rewards)?

We could keep a running sum and count, or, equivalently:

$$Q_{k+1} = Q_k + \frac{1}{k+1} [r_{k+1} - Q_k]$$

This is a common form for update rules:

$$NewEstimate = OldEstimate + StepSize [Target - OldEstimate]$$

Tracking a Nonstationary Problem

Choosing Q_k to be a sample average is appropriate in a stationary problem,

i.e., when none of the $Q^*(a)$ change over time,

But not in a nonstationary problem.

Better in the nonstationary case is:

$$Q_{k+1} = Q_k + \alpha[r_{k+1} - Q_k]$$

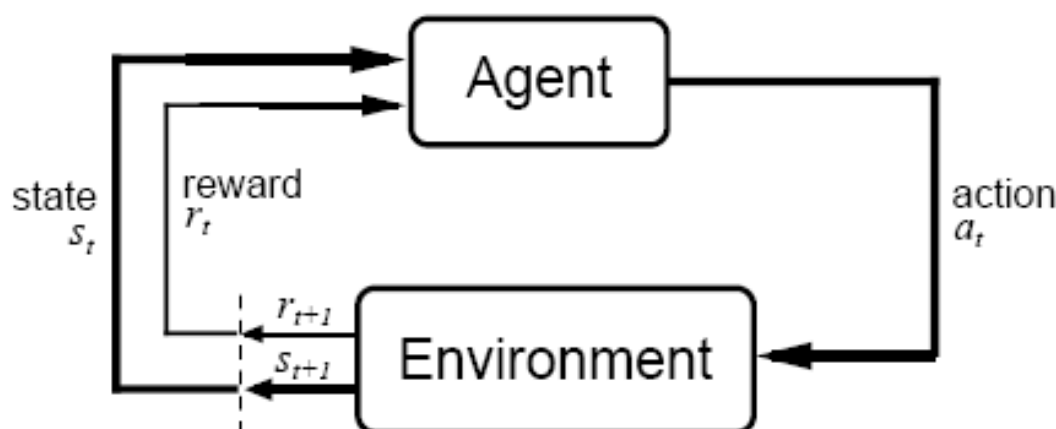
for constant α , $0 < \alpha \leq 1$

$$= (1 - \alpha)^k Q_0 + \sum_{i=1}^k \alpha(1 - \alpha)^{k-i} r_i$$

exponential, recency-weighted average

Reinforcement Learning: Problem Formulation

The Agent-Environment Interface



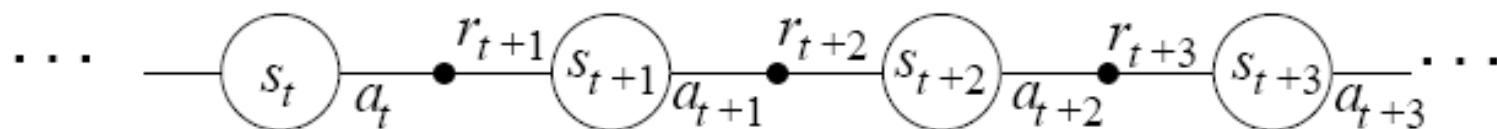
Agent and environment interact at discrete time steps: $t = 0, 1, 2, \dots$

Agent observes state at step t : $s_t \in S$

produces action at step t : $a_t \in A(s_t)$

gets resulting reward: $r_{t+1} \in \mathfrak{R}$

and resulting next state: s_{t+1}



The Agent Learns a Policy

Policy at step t , π_t :

a mapping from states to action probabilities

$\pi_t(s, a)$ = probability that $a_t = a$ when $s_t = s$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
 - Roughly, the agent's goal is to get as much reward as it can over the long run.
-

Goals and Rewards

- ❑ Is a scalar reward signal an adequate notion of a goal?—maybe not, but it is surprisingly flexible.
 - ❑ A goal should specify **what** we want to achieve, not **how** we want to achieve it.
 - ❑ A goal must be outside the agent's direct control—thus outside the agent.
 - ❑ The agent must be able to measure success:
 - explicitly;
 - frequently during its lifespan.
-

Returns

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \dots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step t .

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$R_t = r_{t+1} + r_{t+2} + \dots + r_T,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Returns for Continuing Tasks

Continuing tasks: interaction does not have natural episodes.

Discounted return:

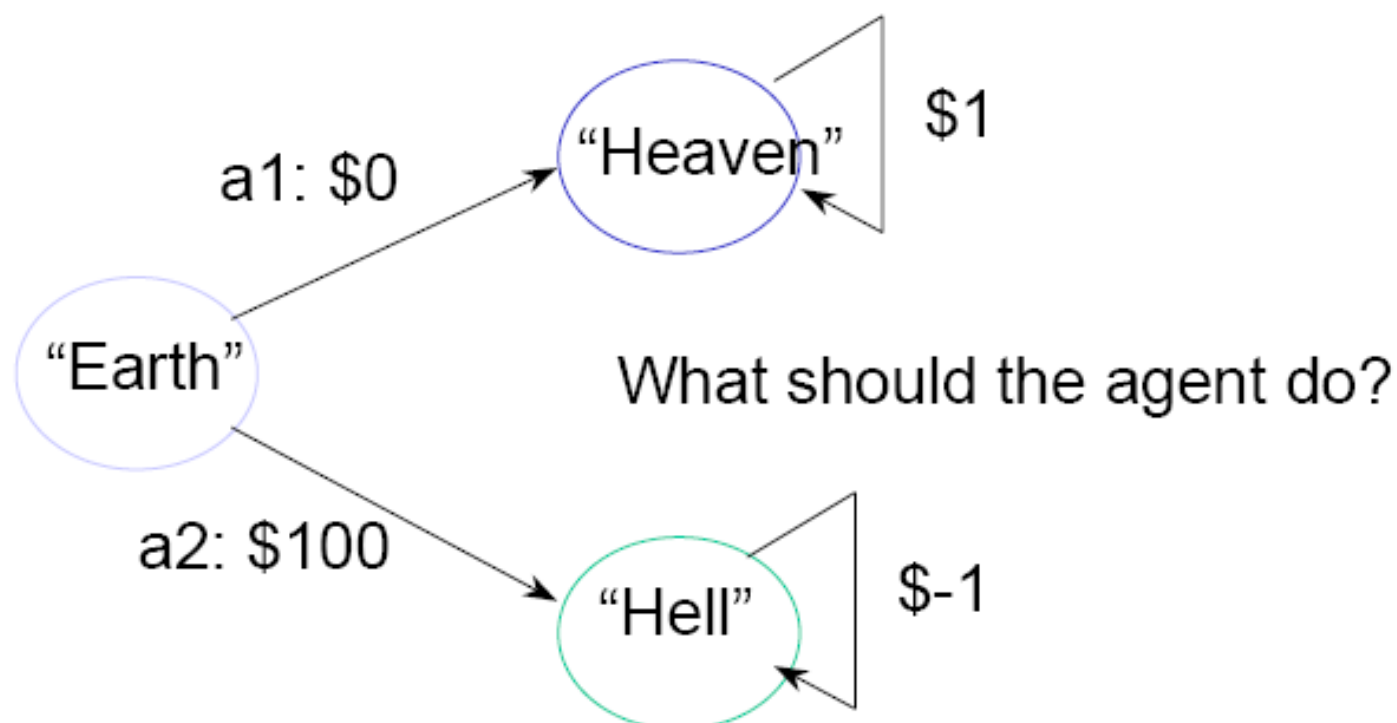
$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where γ , $0 \leq \gamma \leq 1$, is the **discount rate**.

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

Example of MDP

(Schwartz, 1993)



Reward sequence if $a1$ is executed: $0, 1, 1, 1, 1, \dots$

Reward sequence if $a2$ is executed: $100, -1, -1, -1, -1, \dots$

Infinite-horizon -- Discounted Reward

- Graded myopia: “*discounted sum* of rewards”
 - Maximize $\sum_t \gamma^t r_t$
 - Hell if $\gamma < 0.98$, otherwise Heaven
 - Why? (Hint: solve for γ after equating the two infinite reward sums)

Infinite Horizon -- Average Reward

□ Undiscounted: “*average* reward”

- Maximize $\lim_{n \rightarrow \infty} \frac{\sum_{t=1}^n r_t}{n}$

- Go to heaven (why?)

An Example



Avoid **failure**: the pole falling beyond a critical angle or the cart hitting end of track.

As an **episodic task** where episode ends upon failure:

reward = +1 for each step before failure

\Rightarrow return = number of steps before failure

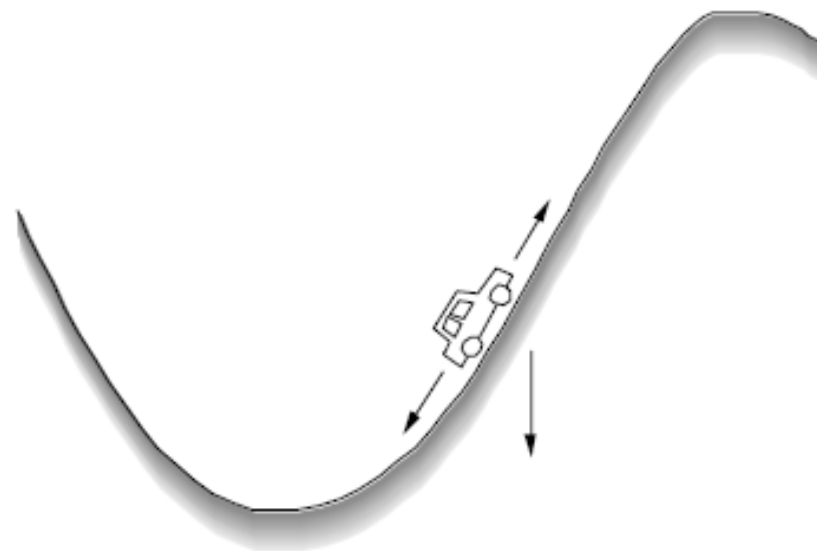
As a **continuing task** with discounted return:

reward = -1 upon failure; 0 otherwise

\Rightarrow return = $-\gamma^k$, for k steps before failure

In either case, return is maximized by avoiding failure for as long as possible.

Another Example



Get to the top of the hill
as quickly as possible.

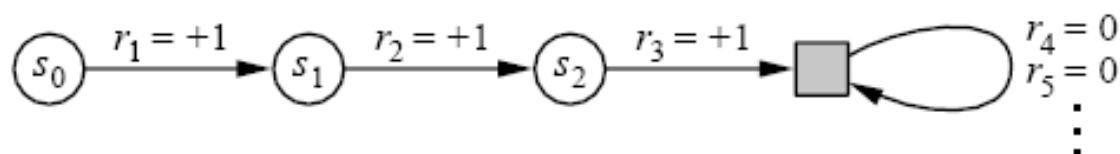
reward = -1 for each step where **not** at top of hill

\Rightarrow return = - number of steps before reaching top of hill

Return is maximized by minimizing
number of steps reach the top of the hill.

A Unified Notation

- ❑ In episodic tasks, we number the time steps of each episode starting from zero.
- ❑ We usually do not have distinguish between episodes, so we write s_t instead of $s_{t,j}$ for the state at step t of episode j .
- ❑ Think of each episode as ending in an absorbing state that always produces reward of zero:



- ❑ We can cover all cases by writing $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$,

where γ can be 1 only if a zero reward absorbing state is always reached.

The Markov Property

- By “the state” at step t , the book means whatever information is available to the agent at step t about its environment.
- The state can include immediate “sensations,” highly processed sensations, and structures built up over time from sequences of sensations.
- Ideally, a state should summarize past sensations so as to retain all “essential” information, i.e., it should have the **Markov Property**:

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0\} = \\ \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\}$$

for all s', r , and histories $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0$.

Markov Decision Processes

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP)**.
- If state and action sets are finite, it is a **finite MDP**.
- To define a finite MDP, you need to give:

- **state and action sets**
- one-step “dynamics” defined by **transition probabilities**:

$$P_{ss'}^a = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \text{ for all } s, s' \in S, a \in A(s).$$

- **reward probabilities**:

$$R_{ss'}^a = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \text{ for all } s, s' \in S, a \in A(s).$$

An Example Finite MDP

Recycling Robot

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
 - Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
 - Decisions made on basis of current energy level: high, low.
 - Reward = number of cans collected
-

Recycling Robot MDP

$S = \{\text{high}, \text{low}\}$

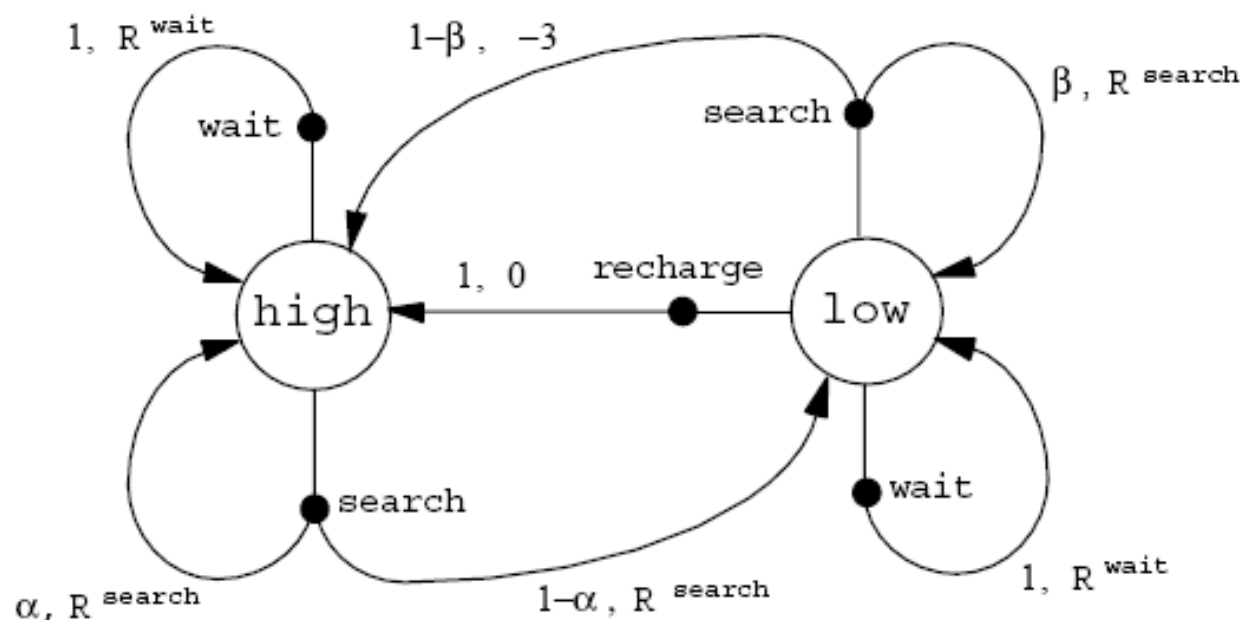
$A(\text{high}) = \{\text{search}, \text{wait}\}$

$A(\text{low}) = \{\text{search}, \text{wait}, \text{recharge}\}$

$R^{\text{search}} =$ expected no. of cans while searching

$R^{\text{wait}} =$ expected no. of cans while waiting

$R^{\text{search}} > R^{\text{wait}}$



Value Functions

- The **value of a state** is the expected return starting from that state; depends on the agent's policy:

State - value function for policy π :

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

- The **value of taking an action in a state under policy π** is the expected return starting from that state, taking that action, and thereafter following π :

Action - value function for policy π :

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

Bellman Equation for a Policy π

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\ &= r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

So:

$$\begin{aligned} V^\pi(s) &= E_\pi \{R_t | s_t = s\} \\ &= E_\pi \{r_{t+1} + \gamma V(s_{t+1}) | s_t = s\} \end{aligned}$$

Or, without the expectation operator:

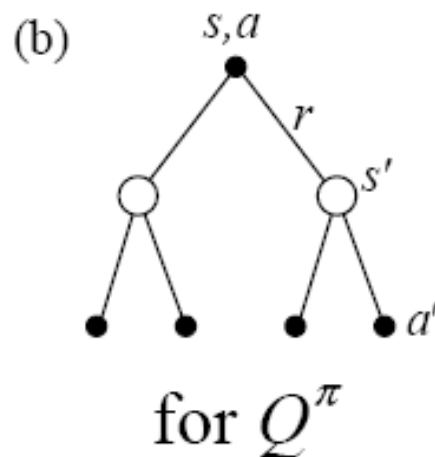
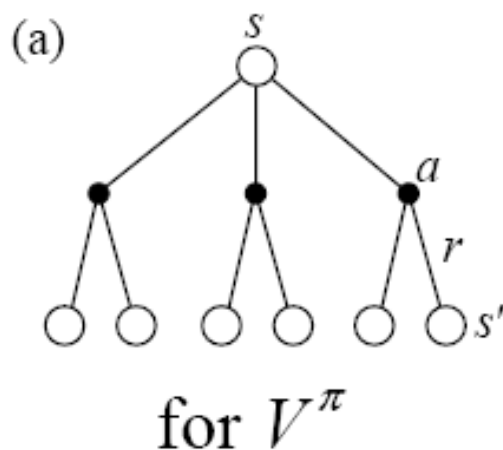
$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]$$

More on the Bellman Equation

$$V^{\pi}(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^{\pi}(s')]$$

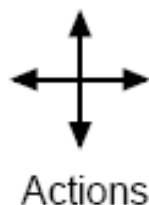
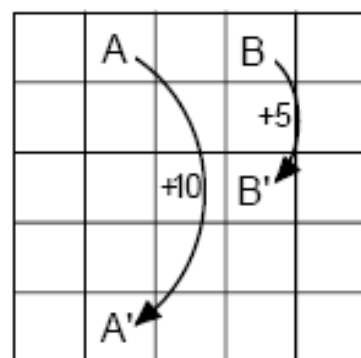
This is a set of equations (in fact, linear), one for each state. The value function for π is its unique solution.

Backup diagrams:



Gridworld

- Actions: north, south, east, west; deterministic.
- If would take agent off the grid: no move but reward = -1
- Other actions produce reward = 0 , except actions that move agent out of special states A and B as shown.



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

State-value function
for equiprobable
random policy;
 $\gamma = 0.9$

Optimal Value Functions

- For finite MDPs, policies can be **partially ordered**:

$$\pi \geq \pi' \quad \text{if and only if} \quad V^\pi(s) \geq V^{\pi'}(s) \quad \text{for all } s \in S$$

- There is always at least one (and possibly many) policies that is better than or equal to all the others. This is an **optimal policy**. We denote them all π^* .

- Optimal policies share the same **optimal state-value function**:

$$V^*(s) = \max_{\pi} V^\pi(s) \quad \text{for all } s \in S$$

- Optimal policies also share the same **optimal action-value function**:

$$Q^*(s, a) = \max_{\pi} Q^\pi(s, a) \quad \text{for all } s \in S \text{ and } a \in A(s)$$

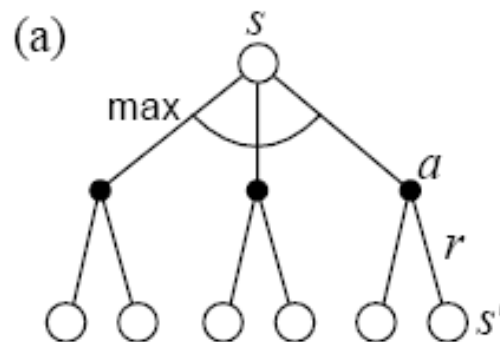
This is the expected return for taking action a in state s and thereafter following an optimal policy.

Bellman Optimality Equation for V^*

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^{\pi^*}(s, a) \\ &= \max_{a \in A(s)} E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \\ &= \max_{a \in A(s)} \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^*(s')] \end{aligned}$$

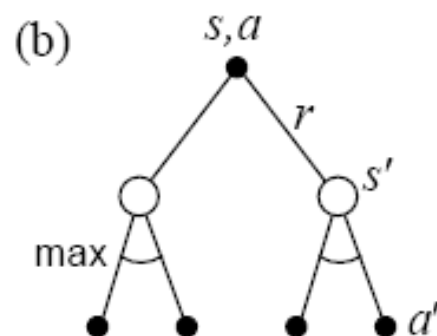
The relevant backup diagram:



Bellman Optimality Equation for Q^*

$$\begin{aligned} Q^*(s, a) &= E \left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a \right\} \\ &= \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma \max_{a'} Q^*(s', a') \right] \end{aligned}$$

The relevant backup diagram:



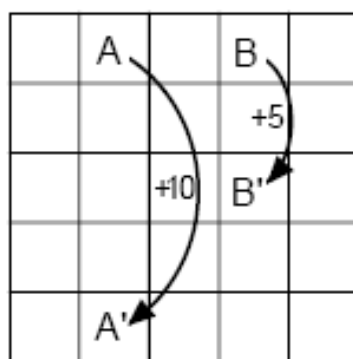
Q^* is the unique solution of this system of nonlinear equations.

Why Optimal State-Value Functions are Useful

Any policy that is greedy with respect to V^* is an optimal policy.

Therefore, given V^* , one-step-ahead search produces the long-term optimal actions.

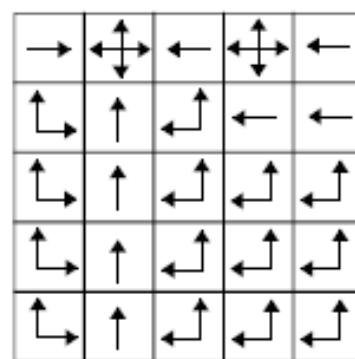
E.g., back to the gridworld:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*



c) π^*