

Outline

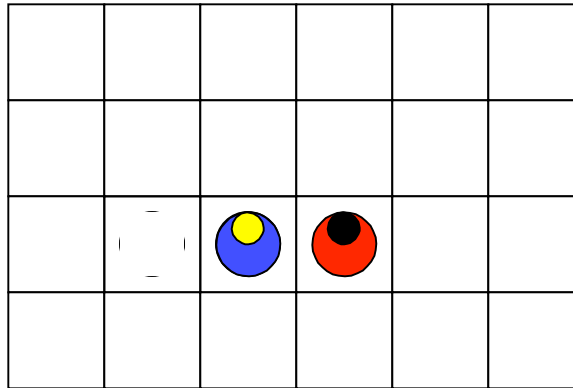
- A. Introduction
- B. Single Agent Learning
 - Markov Decision Processes
 - Regret-Minimizing Algorithms
- C. Game Theory
- D. Multiagent Learning
- E. Future Issues and Open Problems

“Shark” Rules

6x4 grid, two players (with directionality)

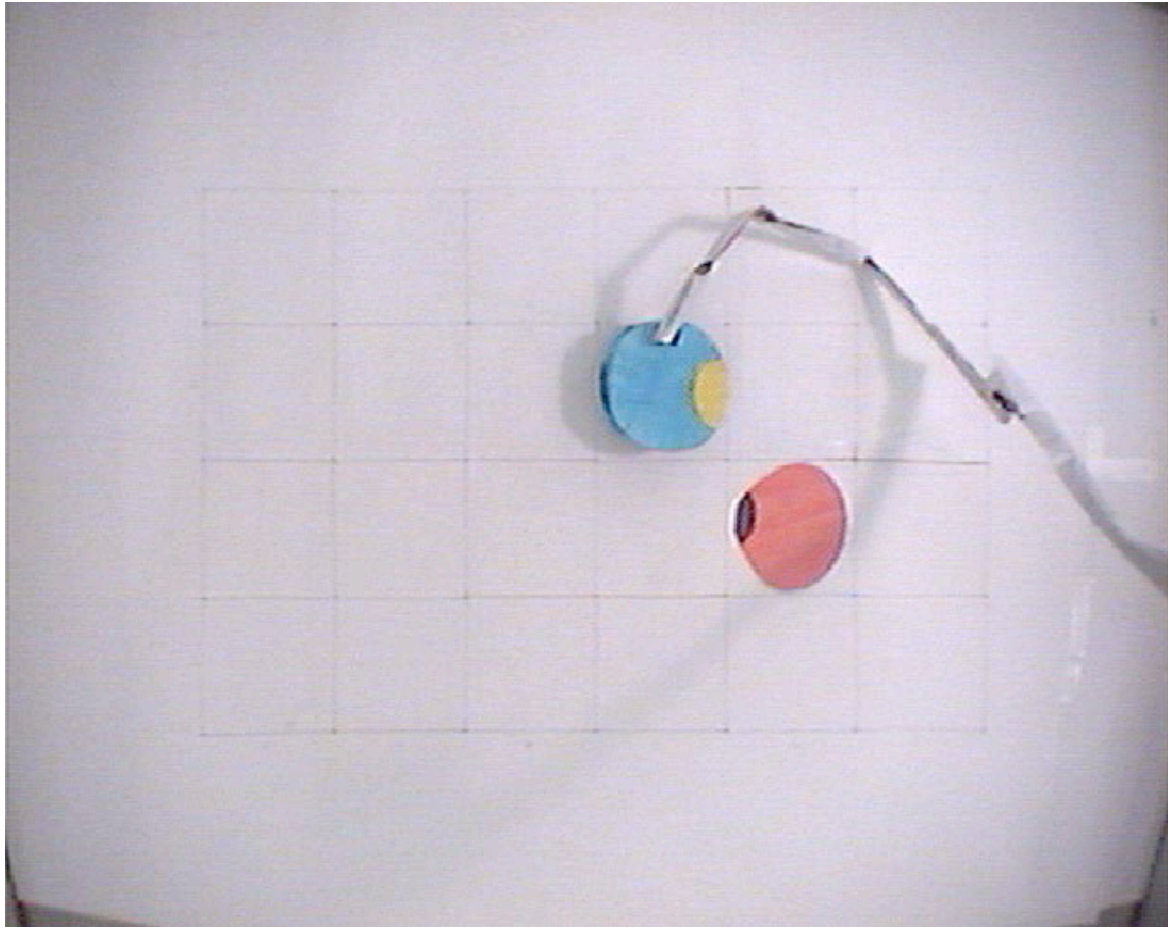
Movement: Step forward, option of turning right or left.

Lose on your turn if facing a wall or your opponent.



SA3-B2

“Sharks” Robot



SA3-B3

Questions

Objective:

- How should the robot act?

Algorithmic:

- How can this be computed?

Learning:

- How can it be learned from experience?

Markov Model Themes

Discrete states: Position on board matters, not history (Markov property).

Finite actions: Agent selects from set of choices (e.g., forward, left 5 degrees, right 5 degrees, stop).

Discrete time: Agent makes turn-taking decisions.

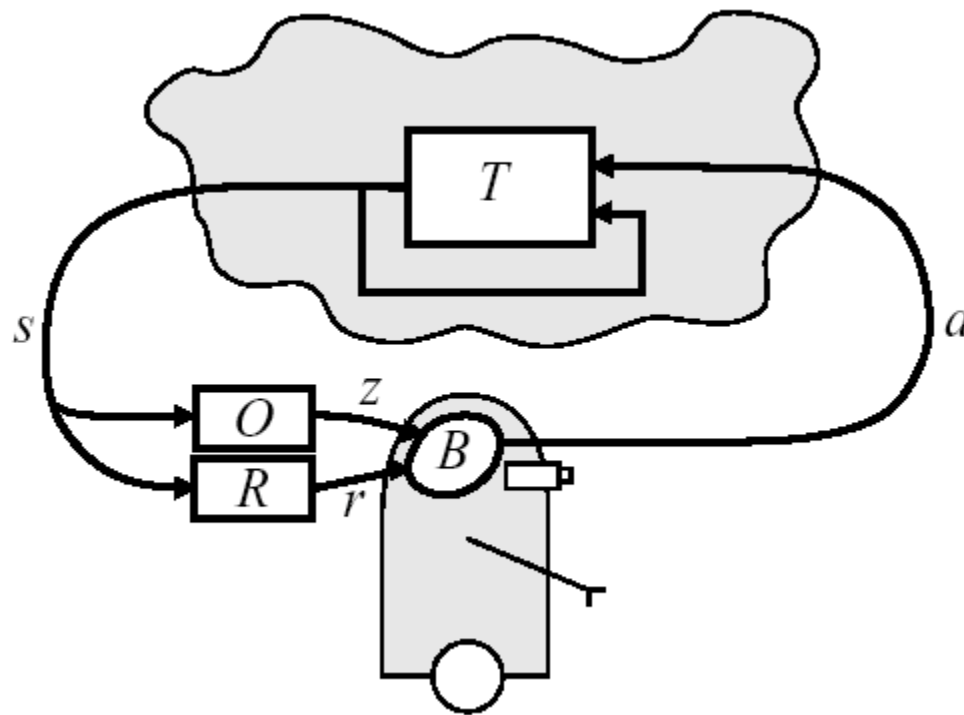
Stochastic transitions: Action (e.g., forward) changes state, but not deterministically (e.g., wheel caught).

Perfect observations: Agent knows the state.

Objective function: Maximize expected return.

Agent Model

The agent's interaction with the environment defined by the transition, observation, and reward functions.












SA3-B6

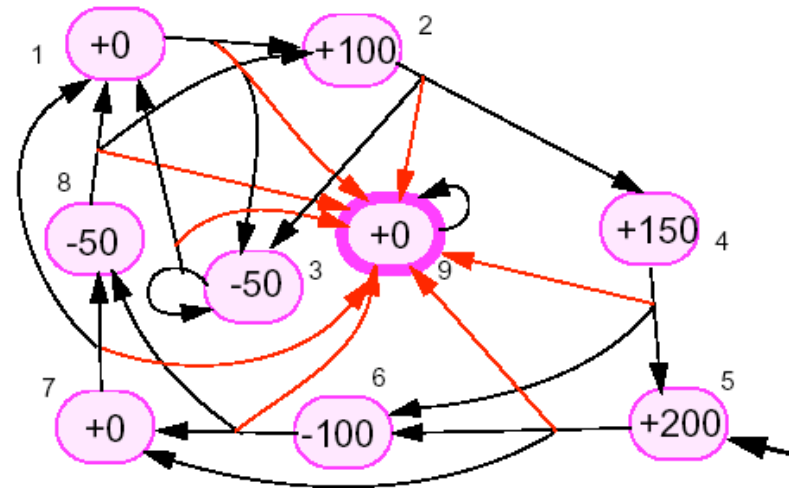
Markov Chains: Definition

Markov models with no action choices: *Markov chains*.

- A finite set of states: S .
- An initial state: $s_0 \in S$.
- A possibly empty set of absorbing states: F .
- A transition function: $T(s, s') = Pr(s' | s)$, the probability of going from state $s \in S$ to state $s' \in S \setminus F$ in one step.
- An additive reward function: $R(s)$.

How They Work: Coinopoly*

| | | |
|--|---|---|
| +0  | +100  | +0  |
| -50  | | +150  |
| +0  | -50  | -100  |
| | | +200  |



Start at “Go”. Flip: heads move one, tails two. Reward based on landing space. “Go to Jail” (“tails” to get out). Before each move, game terminated with probability 0.02.

*Monopoly[®] is a trademark of Parker Brothers.
(Coinopoly isn't.)

State-occupation Probabilities

| t | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 0 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 1 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.49 | 0.49 | 0.00 | 0.02 |
| 2 | 0.24 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.24 | 0.48 | 0.04 |
| 3 | 0.35 | 0.35 | 0.12 | 0.00 | 0.00 | 0.00 | 0.00 | 0.12 | 0.06 |
| 4 | 0.12 | 0.23 | 0.40 | 0.17 | 0.00 | 0.00 | 0.00 | 0.00 | 0.08 |
| 10 | 0.19 | 0.10 | 0.32 | 0.07 | 0.04 | 0.05 | 0.03 | 0.03 | 0.18 |
| 100 | 0.03 | 0.02 | 0.05 | 0.01 | 0.00 | 0.01 | 0.01 | 0.01 | 0.87 |
| 1000 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 |

Long-run reward: +218.1049.

Simulation: Abstract Description

Begin at s_0 with zero accumulated reward.

Set $t = 0$.

1. The agent makes a transition to state s_{t+1} stochastically according to $T(s_t, s_{t+1})$.
2. Increment agent's accumulated reward by $R(s_{t+1})$.
3. Increment t , go to step 1 if $s_t \notin F$.

Examples

Lots of things can be modeled with Markov chains.

- **Queuing systems**: Given probabilities of arrival and service, how long before queue has 10 clients?
- **Genetic algorithms**: How long to find optimum?
- **Statistical sampling**: How longer before “well mixed”?
- **Backgammon**: Given two strategies, how often would player 1 beat player 2?
- **Gambler’s ruin**: How long before broke? Or, probability of earning \$100 before broke?
- **Blackjack**: Expected winnings for a strategy that hits on 18?

Solving Markov Chains: Basics

Several ways to compute value:

- **Search**: Create a tree of possible state sequences, average reward according to its probability.
- **Simulation**: Using weighted sampling, build only a piece of the tree.
- **Dynamic programming**: Calculate the value from all states in S .

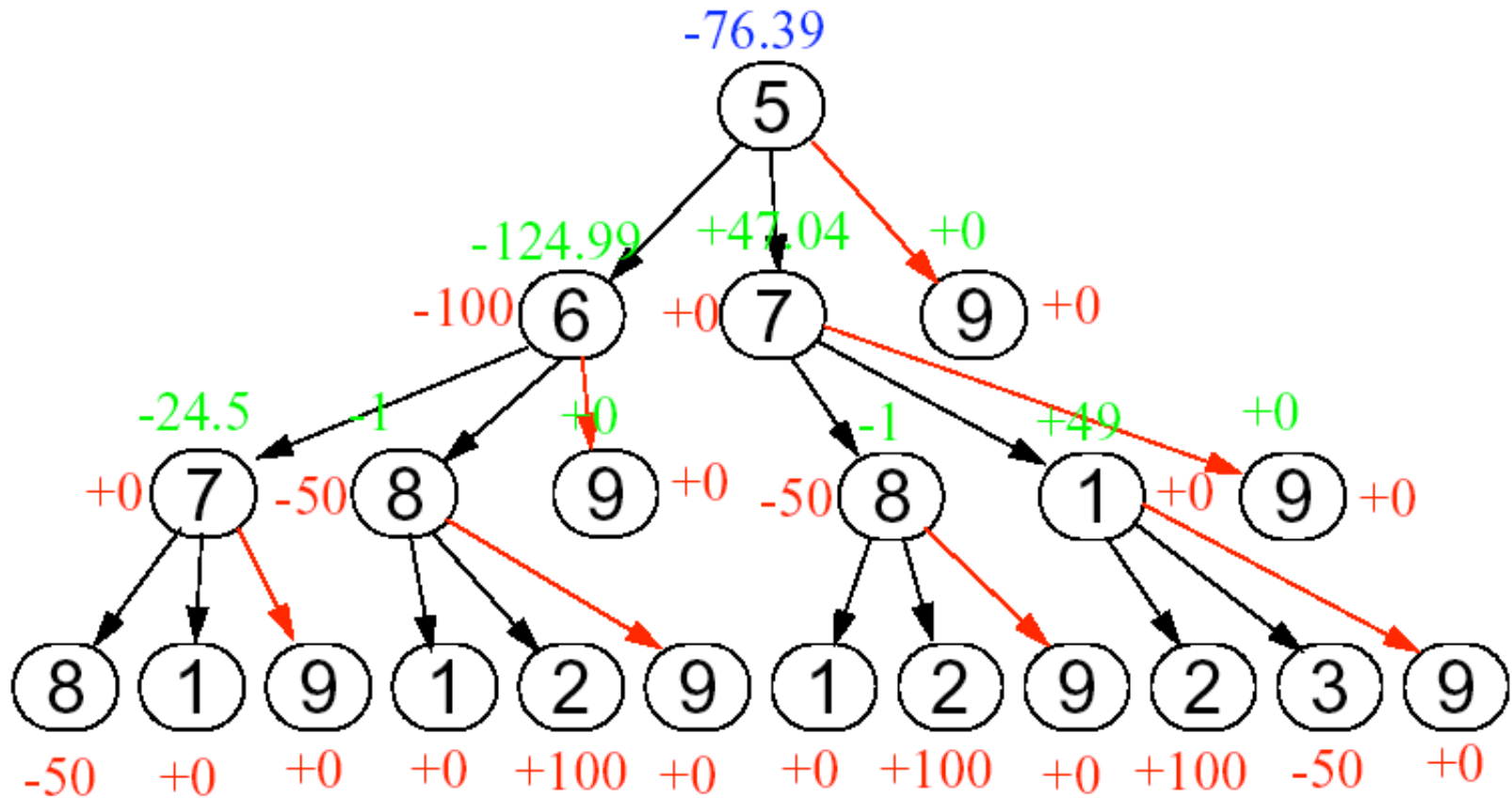
Can be combined in various ways.

Search

Expand out from start state, sum values up the tree.

$$V(s) = R(s) \text{ if } s \in F$$
$$= \min_{s' \in S} T(s, s') (R(s') + V(s')), \text{ otherwise.}$$

Search Example



Returns (truncated) expected value from start state.

Search: Analysis

Strengths:

- Simple.
- Focus computation on reachable states.

Weaknesses:

- May consider the same state multiple times, resulting in a huge search tree.
- Results in approximations for cyclic (infinite horizon) Markov chains.

Simulation

runs: number of simulations to run

sum = 0

for $i = 1$ to *runs* {

$s = s_0$; *runsum* = 0

 while ($s \notin F$) {

$s[i]$ chosen according to $T(s, s[i])$

runsum = *runsum* + $R(s[i])$

$s = s[i]$ }

sum = *sum* + *runsum* }

return *sum* / *runs*

Simulation: Example

Run it 1000 times, take average.

- 5 7 1 3 3 1 3 3 1 3 3 3 1 3 1 3 1 2 4 5 6 8 2 3 9: **-100**
- 5 6 8 1 2 4 5 7 1 3 1 3 9: **+200**
- 5 7 8 2 3 1 3 9: **-50**
- 5 7 8 2 9: **+50**
- 5 6 7 1 3 3 3 3 1 3 1 2 3 3 1 2 9: **-250**
- ...

Mean: +269.65.

Simulation: Analysis

Strengths:

- Simple.
- Focus computation on *likely* reachable states.
- Can get good approximation with little work.
- Applicable without explicit transition probabilities.
(Sampling model.)

Weaknesses:

- Only approximate guarantee.
- May require many samples.
- Somewhat wasteful of data.

Dynamic Programming

Fundamental idea: Instead of solving the problem of finding the value of the start state, we find the value of *all* states (the “value function”).

| | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|------|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 277.41 | 297.65 | 218.49 | 288.96 | 218.10 | 271.60 | 273.51 | 330.78 | 0.00 |

Insight: Because of the linearity of expectations,

$$\begin{aligned}
 V(s) &= R(s) \text{ if } s \in F \\
 &= \sum_{s' \in S} T(s, s') (R(s') + V(s'))
 \end{aligned}$$

System of simultaneous linear equations. Many solution methods.

Dynamic Programming: Analysis

Strengths:

- Calculation exact, even for cyclic Markov chains.
- Calculation is relatively efficient ($O(|S|^3)$).

Weaknesses:

- More complicated to represent and compute values.
- Can “overcompute” in that values are computed for states that don’t matter.

Specific Algorithms

Algorithms in common use combine elements of *search*, *simulation*, and *dynamic programming*.

Value iteration

- Search: Proceeds top down.
- Dynamic programming: Constructs a value function.

Reinforcement learning (TD(0))

- Simulation: Runs trajectories through the space.
- Dynamic programming: Constructs a value function.

Value Iteration

Let $V_0(s) = 0$ for all $s \in S$.

Let $t = 0$.

Repeat:

$$t = t + 1$$

$$V_t(s) = R(s) \text{ for } s \in F$$

$$V_t(s) = \max_{s' \in S} T(s, s') (R(s') + V_{t-1}(s')) \text{ for } s \in S$$

until $(\max_s | V_{t-1}(s) - V_t(s) | < e)$

After a finite number of iterations, $V_t(s) \approx V(s)$.

Reinforcement-learning Methods

How can we use experience to learn the values?

Temporal differences: Adjust value for state s using k steps of observed rewards, followed by estimated value function for last state (if not final):

$$\Delta V(s) = \Delta(r_0 + r_1 + \dots + r_k - V(s_k)).$$

TD(α): Combine estimates, $(1-\alpha)\alpha^k$ for k -step estimate.

Think of α as amount of weight deferred to later steps.

Simulation version called “RTDP” (real-time dynamic programming) (Barto, Bradtke, Singh 95).

TD(1)

Run a complete trajectory (end in a final state).

Keep track of visited states and total rewards.

For each state, train towards observed total reward.

With appropriate learning rate decay, V becomes a more and more accurate prediction over time.

Like simulation method for all visited states.

Supervised training on outcomes, Monte Carlo.

TD(0)

From the Bellman equation,

$$V(s) = \sum_{s'} T(s, s') (R(s') + V(s'))$$

or

$$E_{s'} [R(s') + V(s')] - V(s) = 0$$

So, we train for one-step consistency:

$$\Delta V(s) = \Delta (R(s') + V(s') - V(s)).$$

Connecting TD(0) and TD(1)

Can view TD(1) update (wait until the end), as a sequence of TD(0)-type updates (do it right away).

Trajectory: $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_f$

TD(1): $\Delta V(s_0) = \Delta(r_0 + r_1 + \dots + V(s_f)) - V(s_0)$.

Break TD(1) into steps:

- step 1: $\Delta V(s_0) = \Delta(r_0 + V(s_1)) - V(s_0)$ (TD(0) for s_0)
- step 2: $\Delta V(s_1) = \Delta(r_1 + V(s_2)) - V(s_1)$ (TD(0) for s_1)
- ...

TD(1) is telescoping sum of TD(0)s!

TD(λ)

For *every* state (not just the one just visited):

$$\lambda V(s) = \lambda (r_t + V(s_{t+1})) - V(s_t) e(s)$$

Eligibility $e(s)$ is λ^k where k is steps since s was visited.

When $\lambda=0$, only the most recently visited state (TD(0)).

When $\lambda=1$, all visited states get updated (TD(1)).

Has some nice empirical and theoretical properties.

Learning often best at intermediate values (e.g., $\lambda=0.3$).

Where Are We?

Markov models for representing environments.

Algorithms for computing expected value.

Need to get to agents learning and making decisions...

Decision theory in a nutshell:

Select actions to maximize expected utility (value).

Blackjack: Background

Competing directly against the dealer's fixed strategy.

- Try to get close to 21 without going over.
- Face cards worth 10, ace either as 1 or 11 (usable).
- Two cards to dealer & player. One of dealer's faceup.
- Player "hits" to get an additional card.
- Player "sticks" when satisfied (or stops if goes bust).
- Dealer sticks on any sum of 17 or greater.

Assume infinite deck.

Markov Chain: Reward

Rewards: +1 for winning, -1 for losing, and 0 for a draw.

Dealer's sum is d , value of stopping on p ?

(Ignoring aces for the moment.)

- If $p = \text{bust}$, $R((p, d)) = -1$.
- Else if $d = \text{bust}$, $R((p, d)) = +1$.
- Else if $d \geq 17$ and $d > p$, $R((p, d)) = -1$.
- Else if $d \geq 17$ and $d = p$, $R((p, d)) = 0$.
- Else if $d \geq 17$ and $d < p$, $R((p, d)) = +1$.
- Else if $d < 17$, $R((p, d)) = \sum_{d'} Pr(d' | d) R((p, d'))$

Decisions, Decisions...

Can compute $R((p, d))$ for any p and d .

Create an optimal “stopping” strategy for the player.

For example, should the player hit if the dealer is showing a 5 (no usable ace) and the player a 14 (no usable ace)?

- $R((21,5)) = 0.89,$ $R((16,5)) = \square 0.17$
- $R((20,5)) = 0.67,$ $R((15,5)) = \square 0.17$
- $R((19,5)) = 0.44,$ $R((14,5)) = \square 0.17$
- $R((18,5)) = 0.20,$ $R((13,5)) = \square 0.17$
- $R((17,5)) = \square 0.04,$ $R((12,5)) = \square 0.17$

Expected Value of a Hit

Need to compare value of stick (-0.17) with that of hit.
For one hit, value is the average value for sticking on the resulting number.

- A -0.17 5 0.44 9 -1.00 K -1.00
- 2 -0.17 6 0.67 10 -1.00
- 3 -0.04 7 0.89 J -1.00
- 4 0.20 8 -1.00 Q -1.00

which is -0.32. It is better to stick!

Optimal Strategy

Stick if it's better than hitting. (Duh.)

Value for sticking if dealer has d and player p is $R((p,d))$.

Value of hitting *exactly once* is $\sum_{p'} Pr(p' | p) R((p', d))$.

Easy.

But, might want to hit multiple times...

Multistep: Dynamic Programming

How do we express the value of the optimal strategy?

Optimal values for states are interrelated:

$$V((p, d)) = \max(R((p, d)), \sum_{p'} Pr(p' | p) V((p', d))).$$

Bellman equation: Markov chain equation, with a “max”.

Computing the Optimal Strategy

Given the optimal value function, V , how choose?

One-step lookahead!

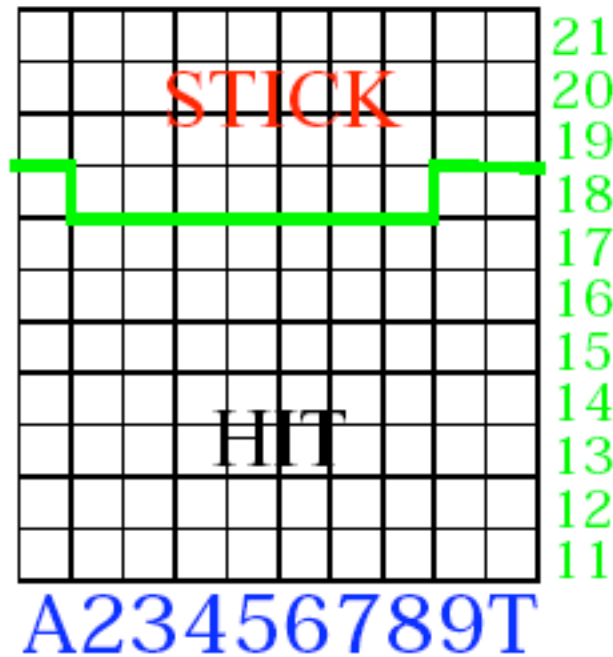
Compare value of sticking to that of hitting once and continuing with an optimal strategy.

Just figure out which component of “max” is largest.

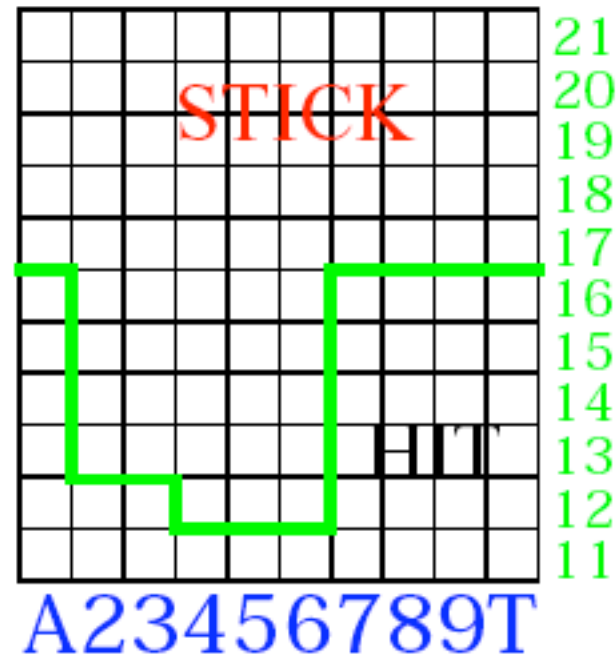
Optimal Blackjack

Decisions based on: current sum (12-21), dealer's faceup card (A-10), whether player has a usable ace.

Usable Ace



No Usable Ace



MDPs: Single Agent Problems

Markov chains let us reason about value in a “dynamical” system where no decisions take place.

Markov decision processes have a single agent, which tries to maximize its expected accumulated reward.

(Bellman 57; Howard 60; Puterman 94)

A further generalization is a zero-sum **Markov game**, in which there are two agents, one tries to maximize reward while the other tries to minimize. (“Sharks” is a deterministic example.) More on this later...

Definition

A Markov decision process (MDP) has:

- A finite set of states: S .
- A finite set of actions: A .
- An initial state: $s_0 \in S$.
- A possibly empty set of absorbing states: F .
- A transition function: $T(s, a, s')$, the probability of going from state $s \in S$ to state $s' \in S \setminus F$ via a in one step.
- An additive reward function: $R(s, a)$.

Sometimes referred to as a “controlled Markov chain.”

Bellman Equations for MDPs

For all $s \in S$,

$$V(s) = \max_a (R(s, a) + \gamma \sum_{s'} T(s, a, s') V(s'))$$

Objective is to choose actions to maximize value:

$$V(s_0) = \max_{a_0 \dots} \mathbb{E} [\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)].$$

A stationary policy will do:

$$\pi(s)$$

Connection to Other Problems

Two common problems can be expressed with MDPs:

- Minimize expected steps to goal:

$R(s,a) = -1$ for all states s other than the goal,

$R(s,a) = 0$ for the goal.

- Maximize probability of reaching the goal:

$R(s,a) = 0$ for all states s , except for an action that takes you into the goal, which has reward 1.

Blackjack: Rewards are zero except for busting (-1) and sticking ($R((p,d))$).

Discounted MDPs

Discount future reward geometrically:

$$V(s_0) = \max_{a_0 \dots} \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right].$$

where $0 < \gamma < 1$ is the discount factor.

Acts like an interest rate, putting more emphasis on short-term gains & losses than on those more distant.

Guarantees optimal value function is well defined.

Other scenarios also well defined (Puterman 94).

Algorithms

MDP Planning (Puterman 94; Boutilier et al. 99)

- value iteration
- policy iteration
- linear programming

Learning (Sutton & Barto 98; Kaelbling et al. 95)

- model-based
- model-free (Q-learning)

Solving MDPs

We can extend the Markov chain algorithms to MDPs:

- **Search**: Add in max nodes.
- **Simulation**: Trickier: what action to choose?
- **Dynamic programming**: Solve Bellman equations.

Bellman equations are not linear, so harder to solve.

Solving the Bellman Equations

Want to compute the value function V .

Once value function is identified, optimal actions can be chosen using one-step lookahead (choose the action that gives the highest expected value).

We'll look at 3 algorithms:

- Value Iteration
- Policy Iteration
- Linear Programming

Value Iteration

Versatile algorithm: replace equality with assignment.

Let $V_0(s) = 0$ for all $s \in S$.

Let $t = 0$.

Repeat:

$$t = t + 1$$

$$V_t(s) = \max_a (R(s, a) + \sum_{s' \in S} T(s, a, s') V_{t-1}(s'))$$

until $(\max_s |V_{t-1}(s) - V_t(s)| < e)$

$\pi(s) = \operatorname{argmax}_a (R(s, a) + \sum_{s'} T(s, a, s') V_t(s'))$: Optimal.

VI in Finite-horizon MDPs

Value iteration maximizes value over fixed horizon h .

$V_h(s)$: maximum value attainable in h steps from s .

We proceed backwards:

- $V_0(s) = 0$: no utility for taking no steps (or final val).
- $V_1(s) = \max_a R(s,a)$: choose action with max utility.
- $V_t(s) = \max_a (R(s, a) + \sum_{s' \in S} T(s, a, s') V_{t-1}(s'))$:
take action with max utility including resulting state.

4x3 Example (Russell and Norvig 94)

| | | | | | |
|--|-------|--|--|----|--|
| | | | | | |
| | | | | +1 | |
| | | | | -1 | |
| | start | | | | |
| | | | | | |

Actions: N, S, E, W. (Hitting wall results in no-op.)

Agent knows initial state, action effects, current state.

Actions 0.8 reliable, 0.1 right angle.

Cost of 0.04 for each action; final payoff (shown in grid).

VI Example

| | | | | | | | | | | | |
|-----------|-------|-------|-------|-----------|-------|-------|-------|-----------|-------|-------|-------|
| 0.000 | 0.000 | 0.000 | +1 | -0.04 | -0.04 | 0.760 | +1 | -0.08 | 0.560 | 0.832 | +1 |
| 0.000 | 0 | 0.000 | -1 | -0.04 | 1 | -0.04 | -1 | -0.08 | 2 | 0.464 | -1 |
| 0.000 | 0.000 | 0.000 | 0.000 | -0.04 | -0.04 | -0.04 | -0.04 | -0.08 | -0.08 | -0.08 | -0.08 |
| 0.392 | 0.738 | 0.890 | +1 | 0.577 | 0.819 | 0.906 | +1 | 0.698 | 0.849 | 0.914 | +1 |
| -0.12 | 3 | 0.572 | -1 | 0.250 | 4 | 0.629 | -1 | 0.472 | 5 | 0.648 | -1 |
| -0.12 | -0.12 | 0.315 | -0.12 | -0.16 | 0.188 | 0.394 | 0.100 | 0.162 | 0.313 | 0.492 | 0.185 |
| 0.809 | 0.868 | 0.918 | +1 | 0.812 | 0.868 | 0.918 | +1 | 0.812 | 0.868 | 0.918 | +1 |
| ... 0.754 | 10 | 0.660 | -1 | ... 0.761 | 15 | 0.660 | -1 | ... 0.762 | >19 | 0.660 | -1 |
| 0.675 | 0.590 | 0.577 | 0.351 | 0.704 | 0.653 | 0.606 | 0.378 | 0.705 | 0.655 | 0.611 | 0.388 |

VI Stuff to Notice

Slow to converge.

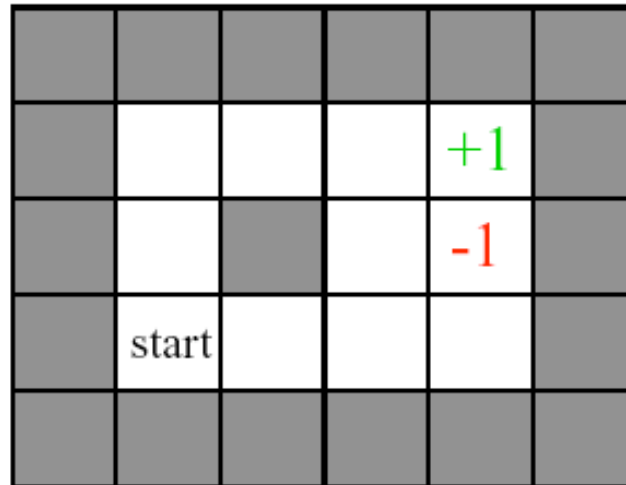
Convergence occurs out from goal.

Information about shortcuts propagates out from goal.

Greedy policy optimal before values completely settle.

Optimal value function is a “fixed policy” of VI.

What if There's No Horizon?



Because of looping, no h is exact value function.

However:

- $V_h(s) \square V(s)$ as h increases.
- Some finite value for h , greedy policy wrt V_h optimal.

Optimal Policy

From optimal value function:

| | | | |
|------------------------|------------------------|------------------------|------------------------|
| B 0.812 → | I 0.868 → | D 0.918 → | +1 |
| H 0.762 ↑ | | G 0.660 ↑ | -1 |
| C 0.705 ↑ | F 0.655 ← | A 0.611 ← | E 0.388 ← |

What's the value of going North from **A**?

Policy Evaluation

From π , can compute V^π , the utility for following π .

Return to the definition of value:

$$V^\pi(s) = R(s, \pi(s)) + \sum_{s'} T(s, \pi(s), s') V^\pi(s')$$

Value is defined recursively (because of looping).

But, choice of action is gone: simple Markov chain!

(Note that if no solution exists, set things to zero.)

Policy Iteration

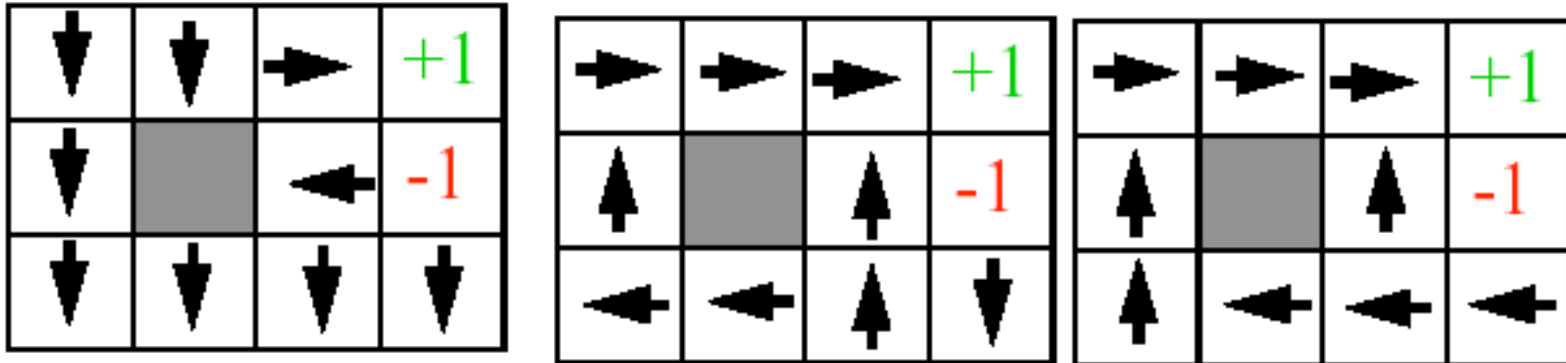
- Start with value function V_0 .
- Let $t = 0$.
- Repeat until V_t optimal:
 - $\pi_{t+1}(s)$ is greedy policy for V_t (policy improvement)
 - $V_{t+1}(s)$ is value of π_{t+1} (policy evaluation)
 - $t = t + 1$

Each policy is an improvement (if possible).

Optimal policy is a fixed point. (Howard 60).

Since set of policies finite, convergence in finite time.

Simulated PI Example



Fewer iterations than VI, but iterations more expensive.

Source of disagreement among practitioners: PI vs. VI.

Open question: Is PI a polynomial-time algorithm?

Speeding Things Up

Note: Value updates can be performed in any order in VI.

Asynchronous updates: use updated values in backup.

Prioritized sweeping: states updated in “priority” order; how much will it change things? (Moore and Atkeson, 1993).

Envelope methods: Solve MDP over reached subset.

Modified PI: PI with evaluation via finite VI. Works well!

Linear Programming

Like solving a system of linear equations, only moreso.

- Linear equations: $Ax = b$.
- Linear programming: $\min_x c x$, s.t. $Ax \geq b$

To express:

$$V(s) = \max_a (R(s, a) + \sum_{s'} T(s, a, s') V(s'))$$

$V(s)$ are variables. “max” is smallest upper bound.

$$\min_{V(s)} \sum_s V(s), \text{ s.t. } V(s') - \sum_s T(s, a, s') V(s) \geq R(s, a)$$

Comparison of Running Times

h is horizon, n states, d is branching factor, m actions

- Value Iteration: $O(h n d m)$.
- Policy Evaluation: $O(n^3)$.
- Policy Improvement: $O(n d m)$.
- Policy Iteration: $O(\min (h (n^3 + n d m), m^n))$. Open!
- Linear programming: Polynomial in number of bits.

Reinforcement Learning in MDPs

Given experience $\langle s, a, r, s' \rangle$ (not what you think)

Model-based RL:

- Keep estimates $R_n(s,a)$, $T_n(s, a, s')$, $n(s,a)$
- $R_n(s,a) += r$; $T_n(s, a, s') += 1$, $n(s,a) += 1$
- Solve MDP $R(s,a) = R_n(s,a)/n(s,a)$; $T(s, a, s') = T_n(s, a, s')/n(s,a)$

Q-learning:

- Keep estimates $Q(s,a)$, learning rate α
- $Q(s,a) += \alpha ((r + \max_{a'} Q(s',a')) - Q(s, a))$
- $Q(s,a)$ approximates $Q(s, a)$

Q-learning Derivation

Rewrite Bellman equation,

$$Q(s, a) = R(s, a) + \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$

or

$$\mathbb{E}_{s'} [R(s, a) + \max_{a'} Q(s', a')] - Q(s, a) = 0$$

So, we train for one-step consistency.

$$\Delta Q(s, a) = \Delta ((r + \max_{a'} Q(s', a')) - Q(s, a))$$

Like VI without a model.

TD(0) well studied; TD(α) version less well understood.

Some Known Results

Model-based RL converges (simple approximation theory)

Q-learning converges if learning rate decayed

appropriately (Watkins & Dayan 92; Tsitsiklis 94; Jaakkola et al. 94; Singh et al. 00)

Both obtain optimal policies quickly, faster than model can be learned! (Kearns & Singh 99)

Careful exploration can lead to polynomial-time approximation algorithms for RL. (Kearns & Singh 98)

Exploration

The exploration-exploitation tradeoff is important in RL.

Tension:

- explore; act to reveal information about environment.
- exploit; act to gain high expected reward.

Q-learning convergence depends only on infinite exploration. Does not hold if greedy actions taken!

□-greedy exploration: Choose random action with prob.

$1 - \epsilon$. Decaying ϵ (GLIE) leads to *policy* convergence.