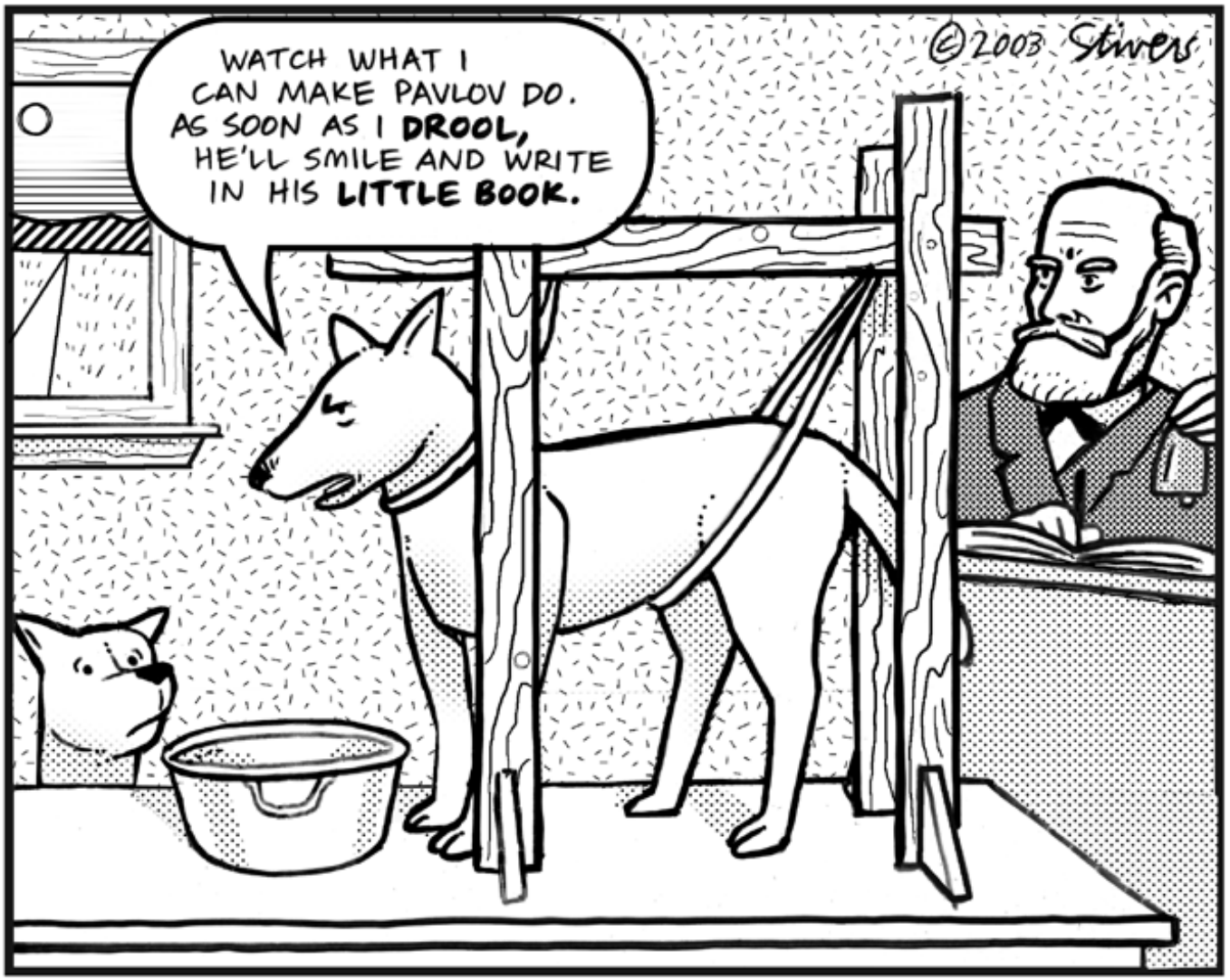
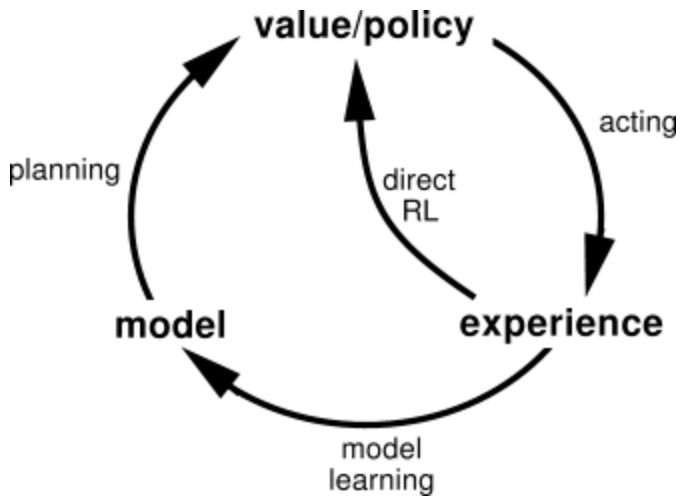
Reinforcement Learning
&
Monte Carlo Planning

(Slides by Alan Fern, Dan Klein, Subbarao Kambhampati, Raj Rao, Lisa Torrey, Dan Weld)


WATCH WHAT I CAN MAKE PAVLOV DO. AS SOON AS I DROOL, HE'LL SMILE AND WRITE IN HIS LITTLE BOOK.
©2003 Stivers

Learning/Planning/Acting



Reinforcement Learning

- Reinforcement learning:
  - Still have an MDP:
    - A set of states $s \in S$
    - A set of actions (per state) A
    - A model T(s,a,s')
    - A reward function R(s,a,s')
  - Still looking for a policy $\pi(s)$
  - New twist: don't know T or R
    - I.e. don't know which states are good or what the actions do
    - Must actually try actions and states out to learn

Main Dimensions

Model-based vs. Model-free

- Model-based vs. Model-free
  - Model-based → Have/learn action models (i.e. transition probabilities)
    - Eg. Approximate DP
  - Model-free → Skip them and directly learn what action to do when (without necessarily finding out the exact model of the action)
    - E.g. Q-learning

Passive vs. Active

- Passive vs. Active
  - Passive: Assume the agent is already following a policy (so there is no action choice to be made; you just need to learn the state values and may be action model)
  - Active: Need to learn both the optimal policy and the state values (and may be action model)

Main Dimensions (contd)

Extent of Backup

- Full DP
  - Adjust value based on values of *all* the neighbors (as predicted by the transition model)
  - Can only be done when transition model is present
- Temporal difference
  - Adjust value based only on the actual transitions observed

Strong or Weak Simulator

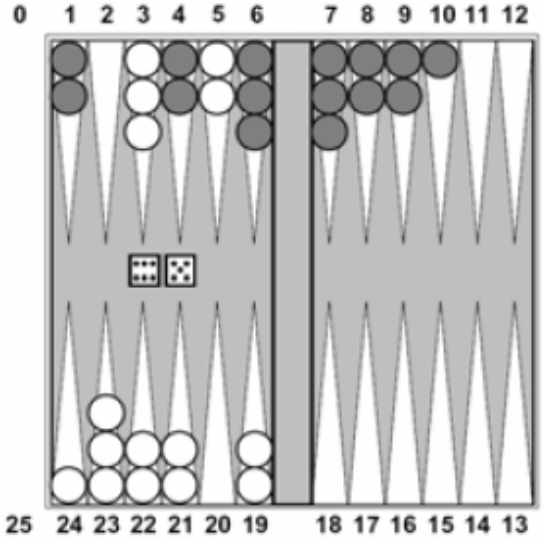
- Strong
  - I can jump to any part of the state space and start simulation there.
- Weak
  - Simulator is the real world and I can teleport to a new state.

Example: Animal Learning

- RL studied experimentally for more than 60 years in psychology
  - Rewards: food, pain, hunger, drugs, etc.
  - Mechanisms and sophistication debated
- Example: foraging
  - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
  - Bees have a direct neural connection from nectar intake measurement to motor planning area

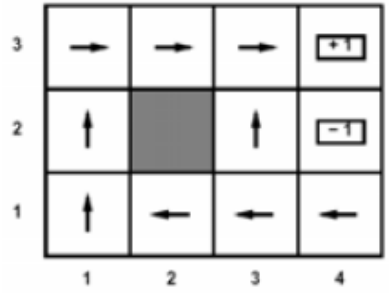
Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to V(s) using a neural network
- Combined with depth 3 search, one of the top 3 players in the world



Does self learning through simulator.
[Infants don't get to "simulate" the world since they neither have T(.) nor R(.) of their world]

Passive Learning



- Simplified task
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You are given a policy $\pi(s)$
  - Goal: learn the state values (and maybe the model)

- In this case:
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We'll get to the general case soon

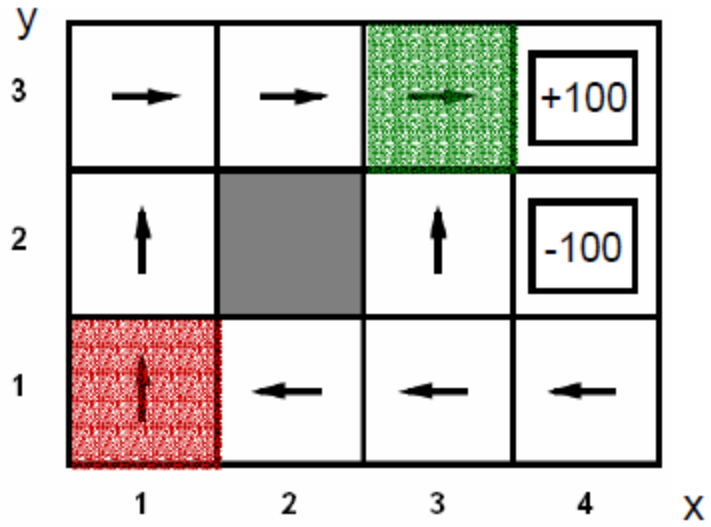
We are basically doing EMPIRICAL Policy Evaluation!

Example: Direct Estimation

- Episodes:

(1,1) up -1
(1,2) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(3,3) right -1
(4,3) exit +100
(done)

(1,1) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(4,2) exit -100
(done)



$\gamma = 1, R = -1$

U(1,1) ~ (92 + -106) / 2 = -7

U(3,3) ~ (99 + 97 + -102) / 3 = 31.3

But we *know* this will be wasteful
(since it misses the correlation between values of neibhoring states!)

Do DP-based policy evaluation!

Model-Based Learning

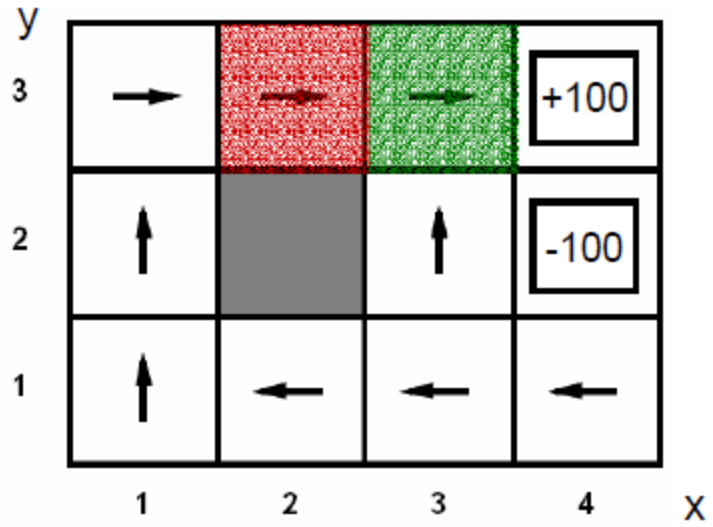
- Idea:
  - Learn the model empirically (rather than values)
  - Solve the MDP as if the learned model were correct
- Empirical model learning
  - Simplest case:
    - Count outcomes for each s,a
    - Normalize to give estimate of T(s,a,s')
    - Discover R(s,a,s') the first time we experience (s,a,s')
  - More complex learners are possible (e.g. if we know that all squares have related action outcomes, e.g. "stationary noise")

Example: Model-Based Learning

- Episodes:

(1,1) up -1
(1,2) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(3,3) right -1
(4,3) exit +100
(done)

(1,1) up -1
(1,2) up -1
(1,3) right -1
(2,3) right -1
(3,3) right -1
(3,2) up -1
(4,2) exit -100
(done)



$\gamma = 1$

T(<3,3>, right, <4,3>) = 1 / 3

T(<2,3>, right, <3,3>) = 2 / 2

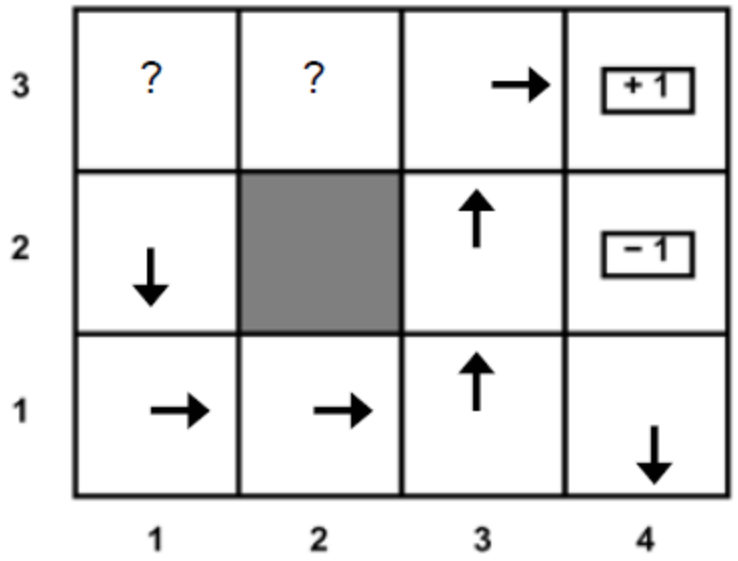
Model-Based Learning

- In general, want to learn the optimal policy, not evaluate a fixed policy

- Idea: adaptive dynamic programming
  - Learn an initial model of the environment:
  - Solve for the optimal policy for this model (value or policy iteration)
  - Refine model through experience and repeat
  - Crucial: we have to make sure we actually learn about all of the model

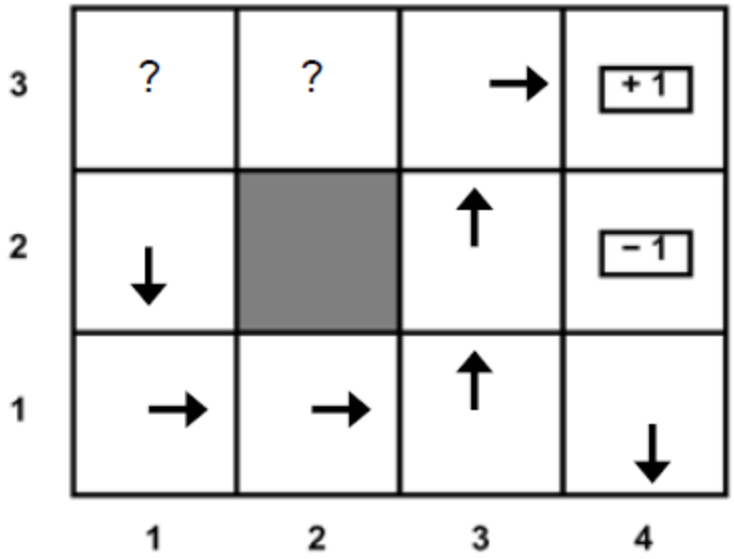
Example: Greedy ADP

- Imagine we find the lower path to the good exit first
- Some states will never be visited following this policy from (1,1)
- We'll keep re-using this policy because following it never collects the regions of the model we need to learn the optimal policy



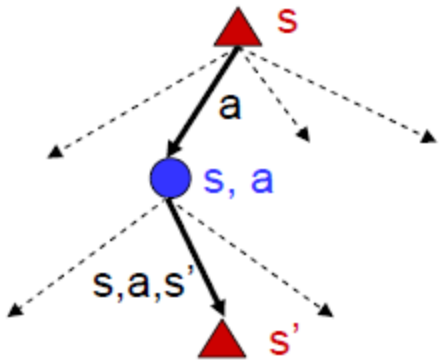
What Went Wrong?

- Problem with following optimal policy for current model:
  - Never learn about better regions of the space if current policy neglects them
- Fundamental tradeoff: exploration vs. exploitation
  - Exploration: must take actions with suboptimal estimates to discover new rewards and increase eventual utility
  - Exploitation: once the true optimal policy is learned, exploration reduces utility
  - Systems must explore in the beginning and exploit in the limit



Model-Free Learning

- Big idea: why bother learning T?
  - Update each time we experience a transition
  - Frequent outcomes will contribute more updates (over time)
- Temporal difference learning (TD)
  - Policy still fixed!
  - Move values toward value of whatever successor occurs



$$V^{\pi}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, a, s') + \gamma V^{\pi}(s')]$$

$$sample = R(s, a, s') + \gamma V^{\pi}(s')$$

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$$

updated estimate learning rate

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers ($x_1, x_2, x_3, \ldots$)
  - E.g. to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i$$

average of n+1 samples

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers ($x_1, x_2, x_3, \ldots$)
  - E.g. to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n+1}\left(x_{n+1} - \frac{1}{n}\sum_{i=1}^{n} x_i\right)$$

average of n+1 samples

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers ($x_1, x_2, x_3, \ldots$)
  - E.g. to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n+1}\left(x_{n+1} - \frac{1}{n}\sum_{i=1}^{n} x_i\right)$$

$$= \hat{X}_n + \frac{1}{n+1}\left(x_{n+1} - \hat{X}_n\right)$$

average of n+1 samples

learning rate

sample n+1

- Given a new sample x_{n+1}, the new mean is the old estimate (for n samples) plus the weighted difference between the new sample and old estimate

Temporal Difference Learning

- TD update for transition from s to s':

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(R(s) + \gamma V^{\pi}(s') - V^{\pi}(s))$$

updated estimate

learning rate

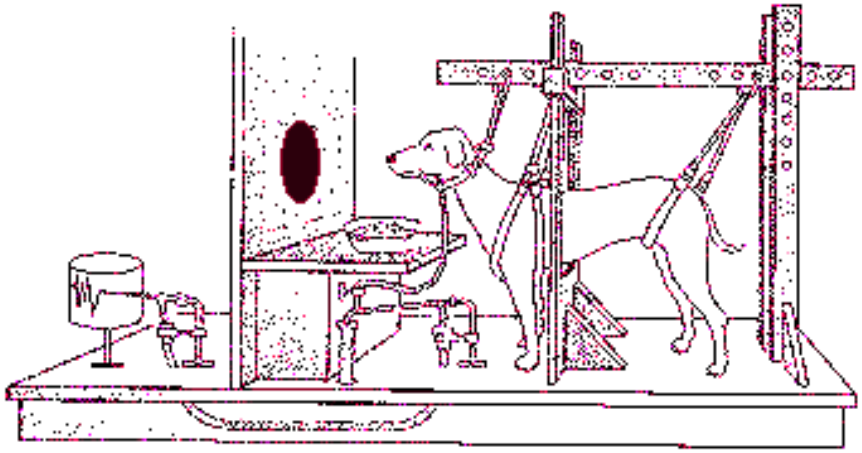
(noisy) sample of value at s based on next state s'

- So the update is maintaining a "mean" of the (noisy) value samples
- If the learning rate decreases appropriately with the number of samples (e.g. 1/n) then the value estimates will converge to true values! (non-trivial)

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

Early Results: Pavlov and his Dog

- Classical (Pavlovian) conditioning experiments
- Training: Bell →Food
- After: Bell → Salivate
- Conditioned stimulus (bell) predicts future reward (food)



(http://employees.csbsju.edu/tcreed/pb/pdoganim.html)

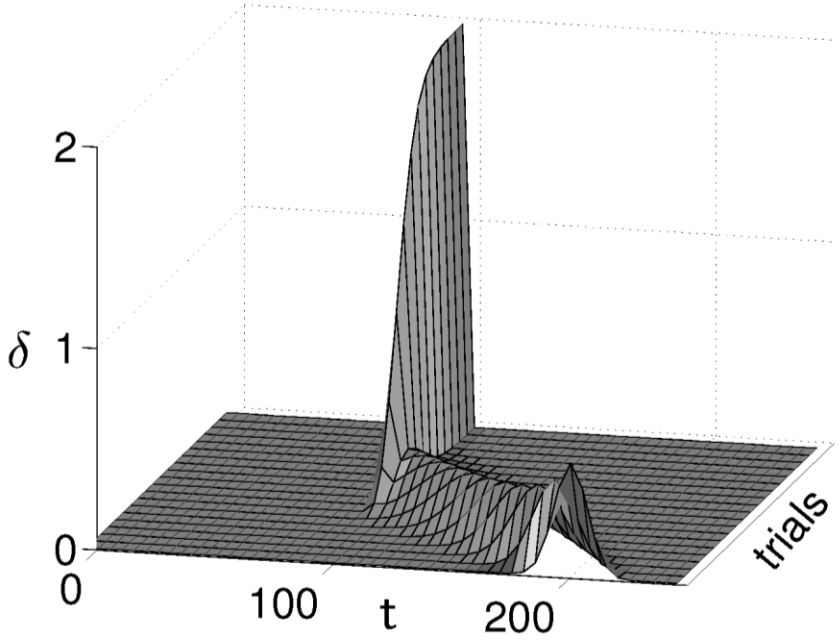
Predicting Delayed Rewards

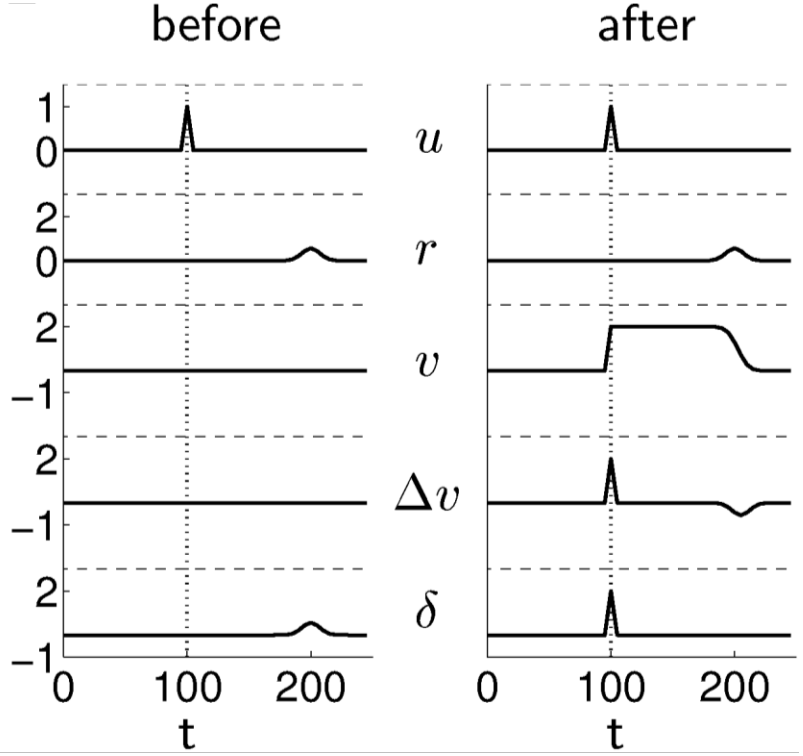
- Reward is typically delivered at the end (when you know whether you succeeded or not)
- Time: $0 \leq t \leq T$ with stimulus $u(t)$ and reward $r(t)$ at each time step t (Note: $r(t)$ can be zero at some time points)
- Key Idea: Make the output $v(t)$ predict *total expected future reward* starting from time t

$$v(t) \approx \left\langle \sum_{\tau=0}^{T-t} r(t+\tau) \right\rangle$$

Predicting Delayed Reward: TD Learning

Stimulus at t = 100 and reward at t = 200



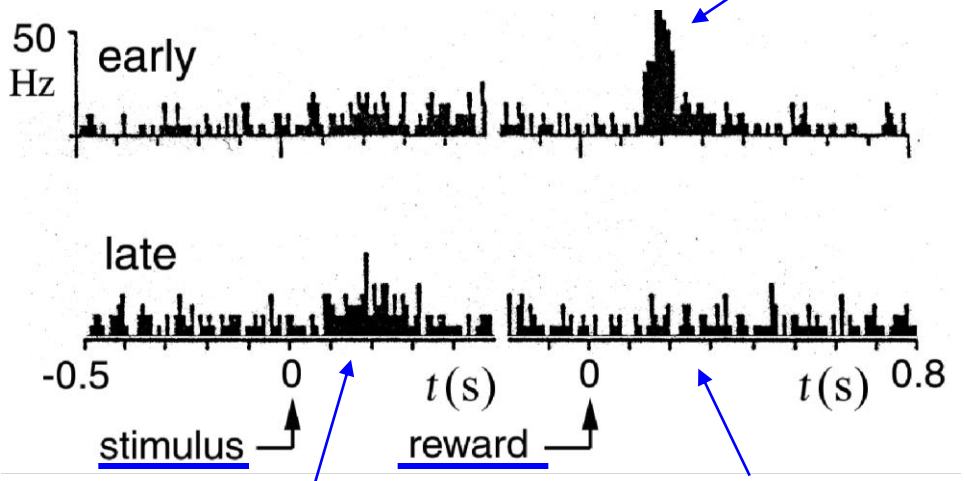


Prediction error δ for each time step
(over many trials)

Prediction Error in the Primate Brain?

Dopaminergic cells in Ventral Tegmental Area (VTA)

Reward Prediction error? $[r(t)+v(t+1)-v(t)]$



Before Training

After Training

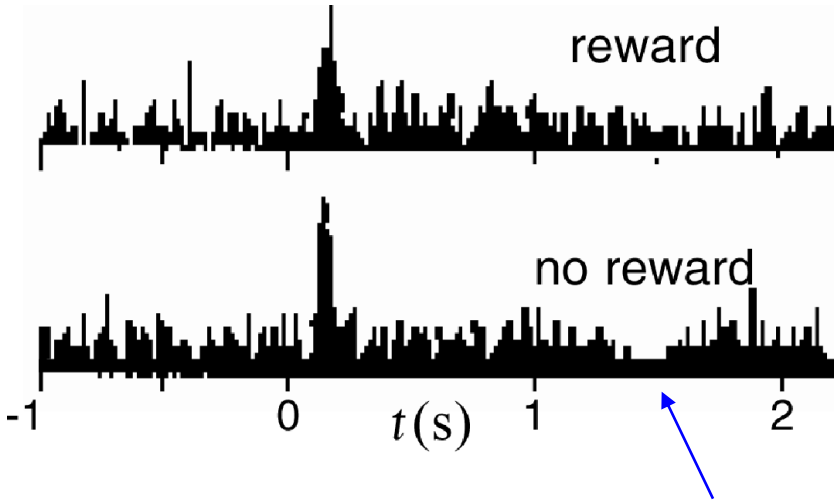
No error

$[0+v(t+1)-v(t)]$

$v(t) \approx r(t)+v(t+1)$

More Evidence for Prediction Error Signals

Dopaminergic cells in VTA



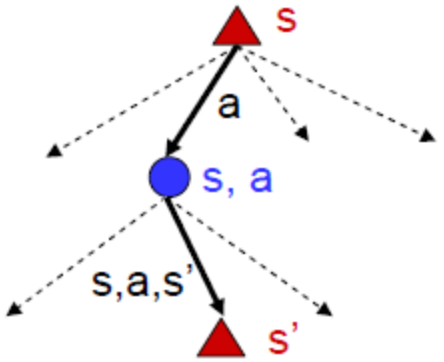
Negative error

$$r(t) = 0, v(t+1) = 0$$

$$[r(t) + v(t+1) - v(t)] = -v(t)$$

Problems with TD Value Learning

- TD value leaning is model-free for policy evaluation
- However, if we want to turn our value estimates into a policy, we're sunk:



$$\pi(s) = \arg\max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

- Idea: learn Q-values directly
- Makes action selection model-free too!

Q-Learning

- Learn Q*(s,a) values
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

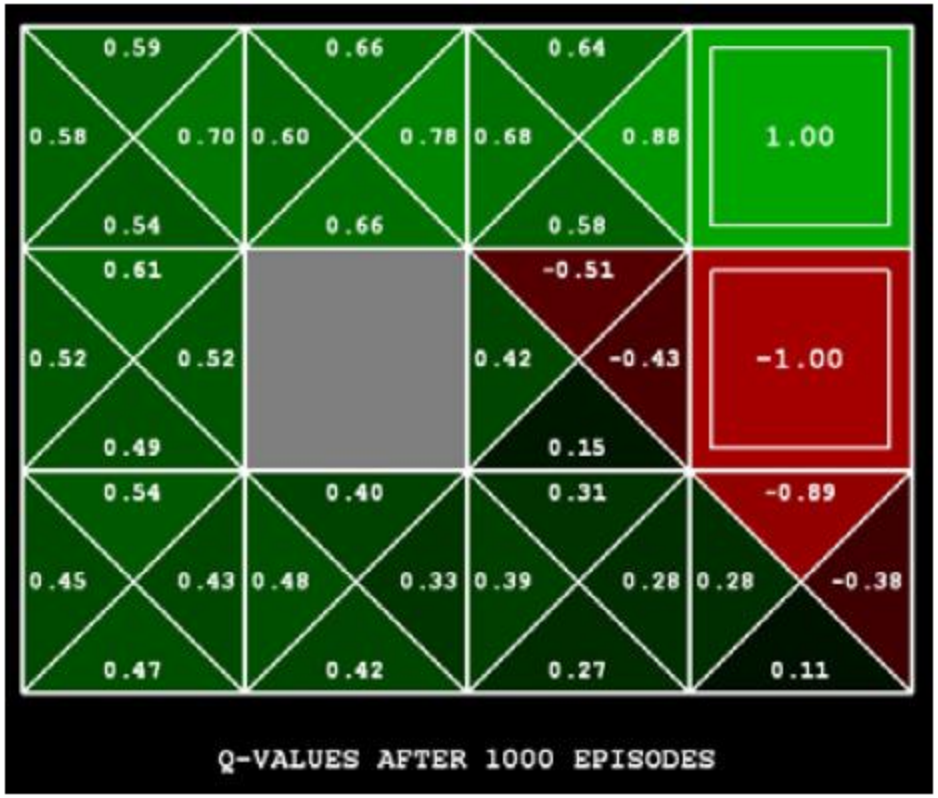
$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Nudge the old estimate towards the new sample:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[sample - Q(s,a) \right]$$

Q-Learning

- Q-learning produces tables of q-values:



Q-Learning Properties

- Will converge to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
- Under certain conditions:
  - The environment model doesn't change
  - States and actions are finite
  - Rewards are bounded
  - Learning rate decays with visits to state-action pairs
    - but not too fast decay. ($\sum_i \alpha(s,a,i) = \infty$, $\sum_i \alpha^2(s,a,i) < \infty$)
  - Exploration method would guarantee infinite visits to every state-action pair over an infinite training period

Exploration / Exploitation

- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With probability ε, act randomly
    - With probability 1-ε, act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - One solution: lower ε over time
    - Another solution: exploration functions

Explore/Exploit Policies

- GLIE Policy 2: Boltzmann Exploration
  - Select action a with probability,

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

  - T is the temperature. Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically start with large T and decrease with time

Model based vs. Model Free RL

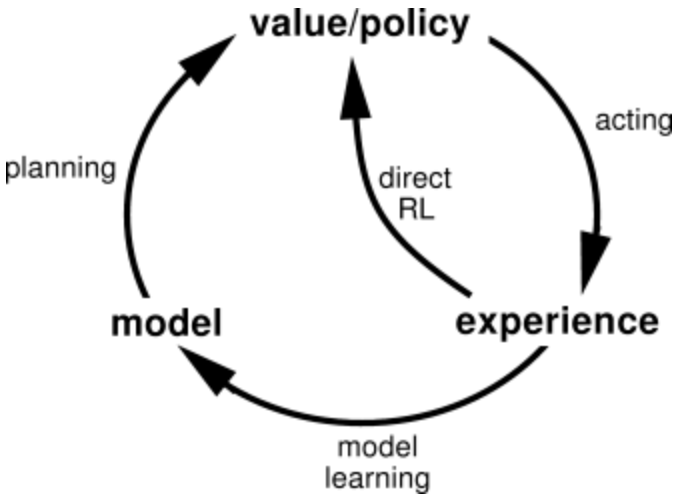
- Model based
  - estimate O($|\mathcal{S}|^2|\mathcal{A}|$) parameters
  - requires relatively larger data for learning
  - can make use of background knowledge easily

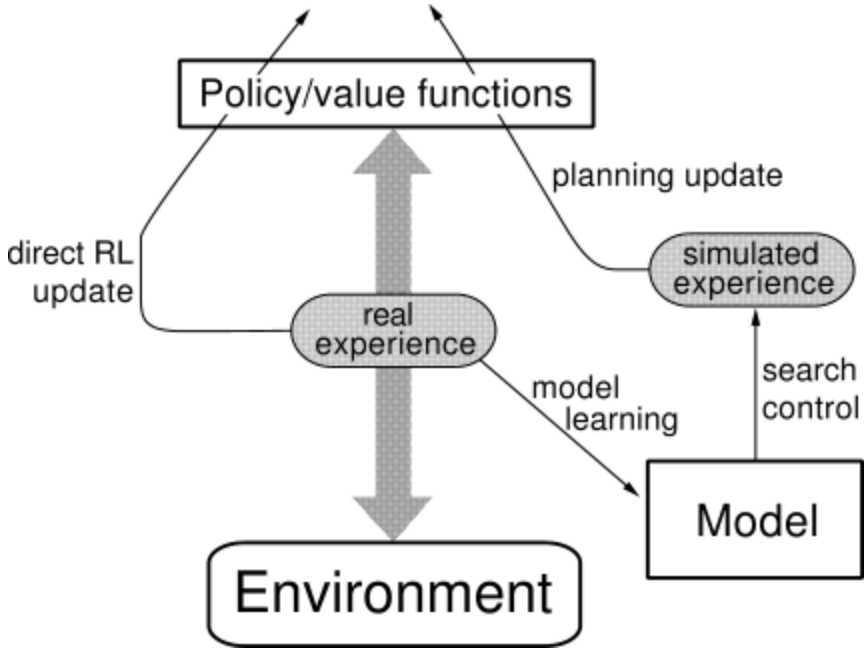
- Model free
  - estimate O($|\mathcal{S}||\mathcal{A}|$) parameters
  - requires relatively less data for learning

Applications of RL

- Games
  - Backgammon, Solitaire, Real-time strategy games
- Elevator Scheduling
- Stock investment decisions
- Chemotherapy treatment decisions
- Robotics
  - Navigation, Robocup
  - http://www.youtube.com/watch?v=CIF2SBVY-J0
  - http://www.youtube.com/watch?v=5FGVgMsiv1s
  - http://www.youtube.com/watch?v=W_gxLKSsSIE
- Helicopter maneuvering

Learning/Planning/Acting



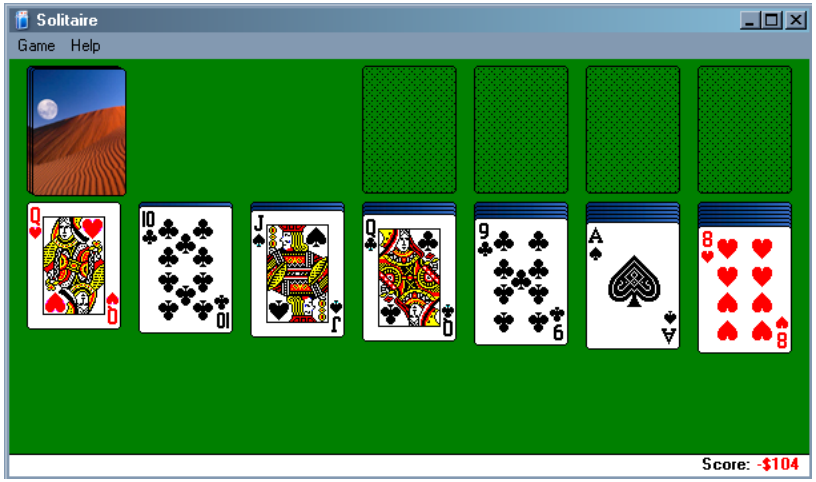


Planning **Monte-Carlo Planning** **Reinforcement Learning**

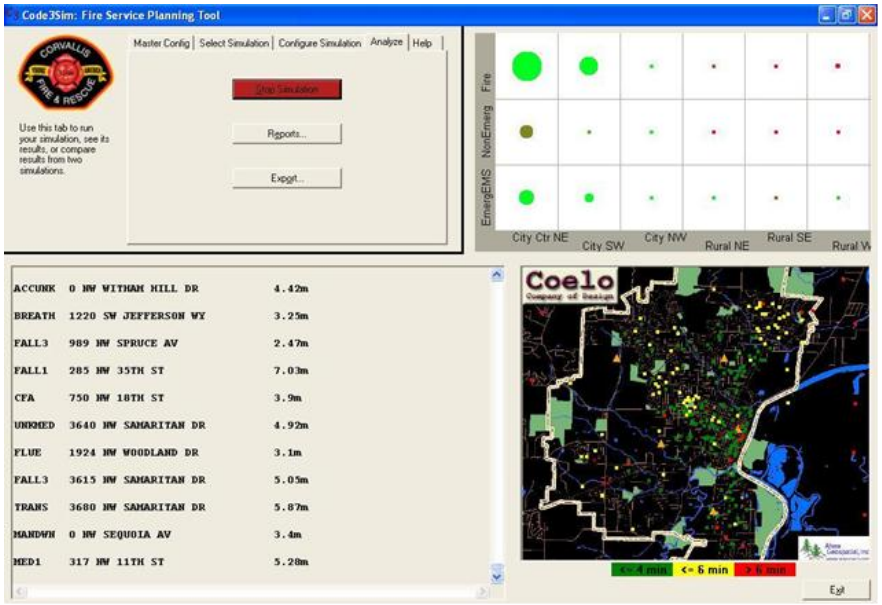
Monte-Carlo Planning

- Often a simulator of a planning domain is available or can be learned from data
  - Even when domain can't be expressed via MDP language

Klondike Solitaire



Fire & Emergency Response



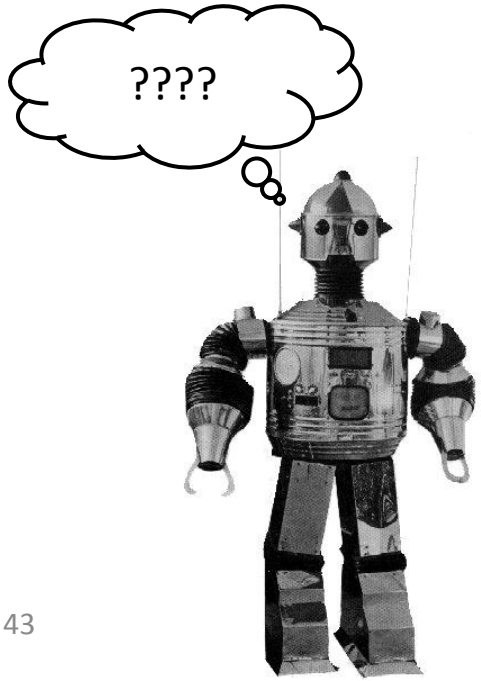
41

Example Domains with Simulators

- Traffic simulators
- Robotics simulators
- Military campaign simulators
- Computer network simulators
- Emergency planning simulators
  - large-scale disaster and municipal
- Sports domains (Madden Football)
- Board games / Video games
  - Go / RTS

In many cases Monte-Carlo techniques yield state-of-the-art performance. Even in domains where model-based planner is applicable.

Slot Machines as MDP?

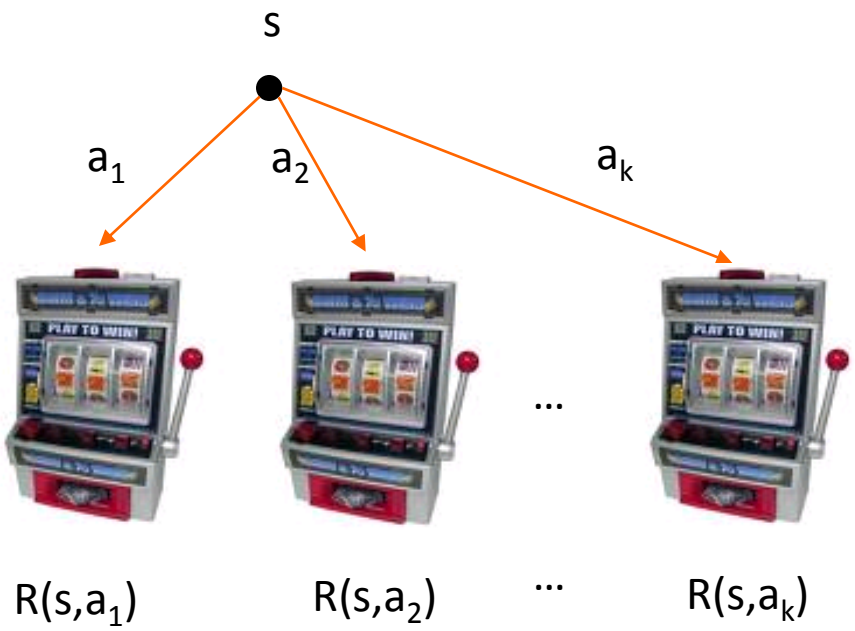


Outline

- Uniform Sampling
  - PAC Bound for Single State MDPs
  - Policy Rollouts for full MDPs
- Adaptive Sampling
  - UCB for Single State MDPs
  - UCT for full MDPs

Single State Monte-Carlo Planning

- Suppose MDP has a single state and k actions
  - Figure out which action has best expected reward
  - Can sample rewards of actions using calls to simulator
  - Sampling a is like pulling slot machine arm with random payoff function R(s,a)

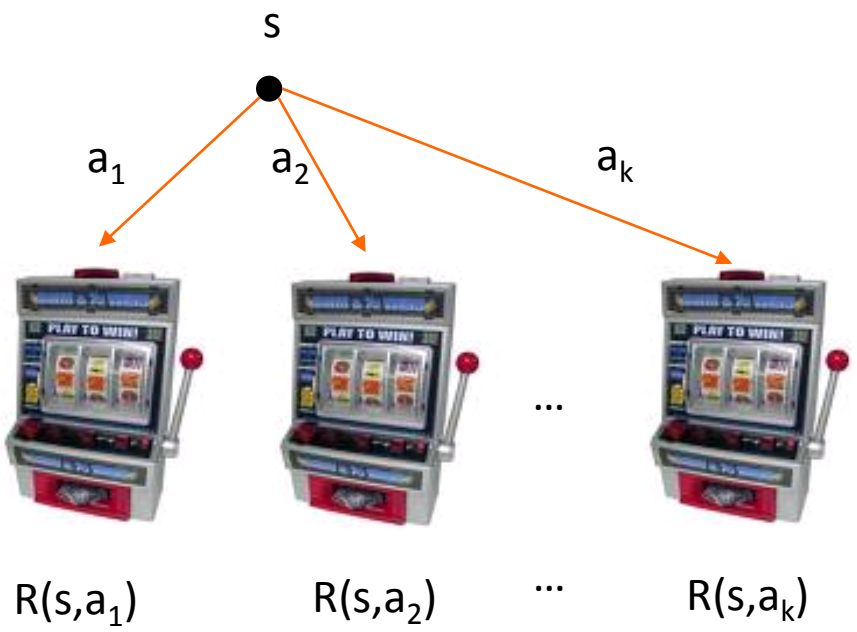


Multi-Armed Bandit Problem

PAC Bandit Objective

Probably Approximately Correct (PAC)

- Select an arm that probably (w/ high probability, 1-δ) has approximately (i.e., within ε) the best expected reward
- Use as few simulator calls (or pulls) as possible

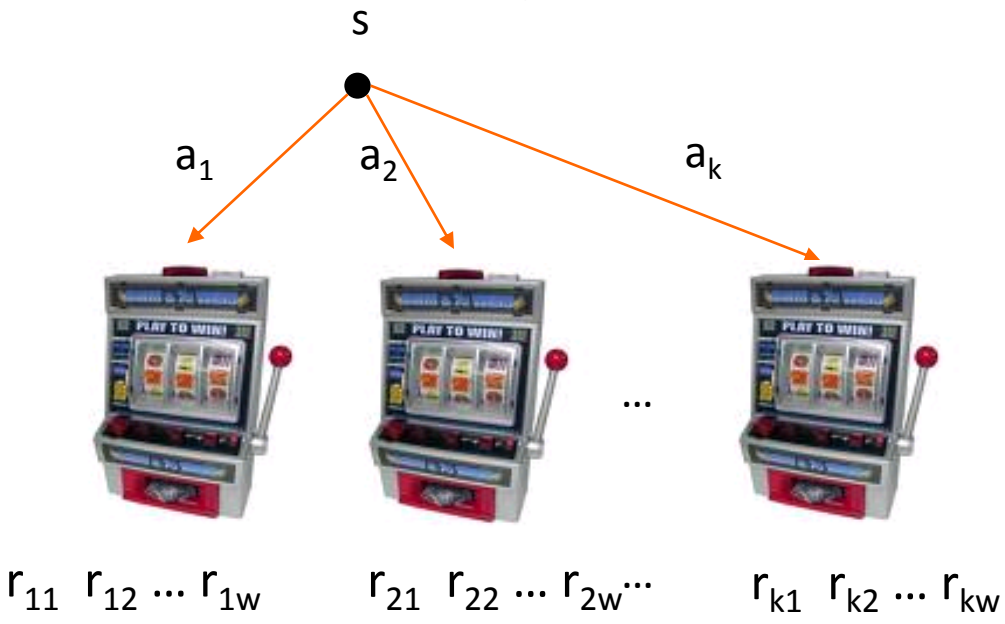


Multi-Armed Bandit Problem

UniformBandit Algorithm

NaiveBandit from [Even-Dar et. al., 2002]

1. Pull each arm **w** times (uniform pulling).
2. Return arm with best average reward.



How large must w be to provide a PAC guarantee?

Aside: Additive Chernoff Bound

- Let R be a random variable with maximum absolute value Z. An let r_i (for i=1,…,w) be i.i.d. samples of R
- The Chernoff bound gives a bound on the probability that the average of the r_i are far from E[R]

Chernoff Bound

$$\Pr\left(\left|E[R]-\frac{1}{w}\sum_{i=1}^{w} r_i\right| \geq \varepsilon\right) \leq \exp\left(-\left(\frac{\varepsilon}{Z}\right)^2 w\right)$$

Equivalently:

With probability at least $1-\delta$ we have that,

$$\left|E[R]-\frac{1}{w}\sum_{i=1}^{w} r_i\right| \leq Z\sqrt{\frac{1}{w}\ln\frac{1}{\delta}}$$

UniformBandit PAC Bound

With a bit of algebra and Chernoff bound we get:

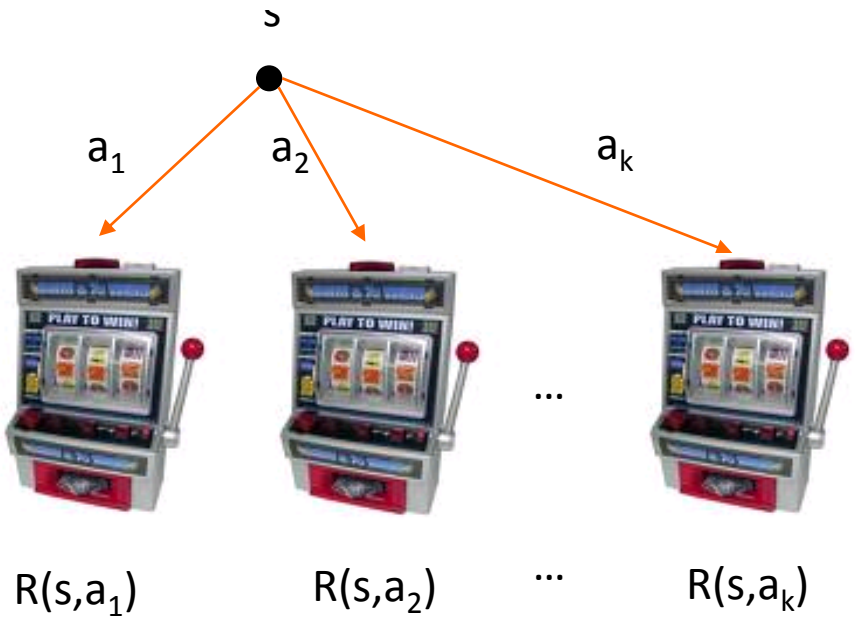
If $w \geq \left(\frac{R_{\max}}{\varepsilon} \right)^2 \ln \frac{k}{\delta}$ for all arms simultaneously

$$\left| E[R(s,a_i)] - \frac{1}{w} \sum_{j=1}^{w} r_{ij} \right| \leq \varepsilon$$

with probability at least $1-\delta$

- That is, estimates of all actions are **ε**–accurate with probability at least 1- δ
- Thus selecting estimate with highest value is approximately optimal with high probability, or PAC

Simulator Calls for UniformBandit



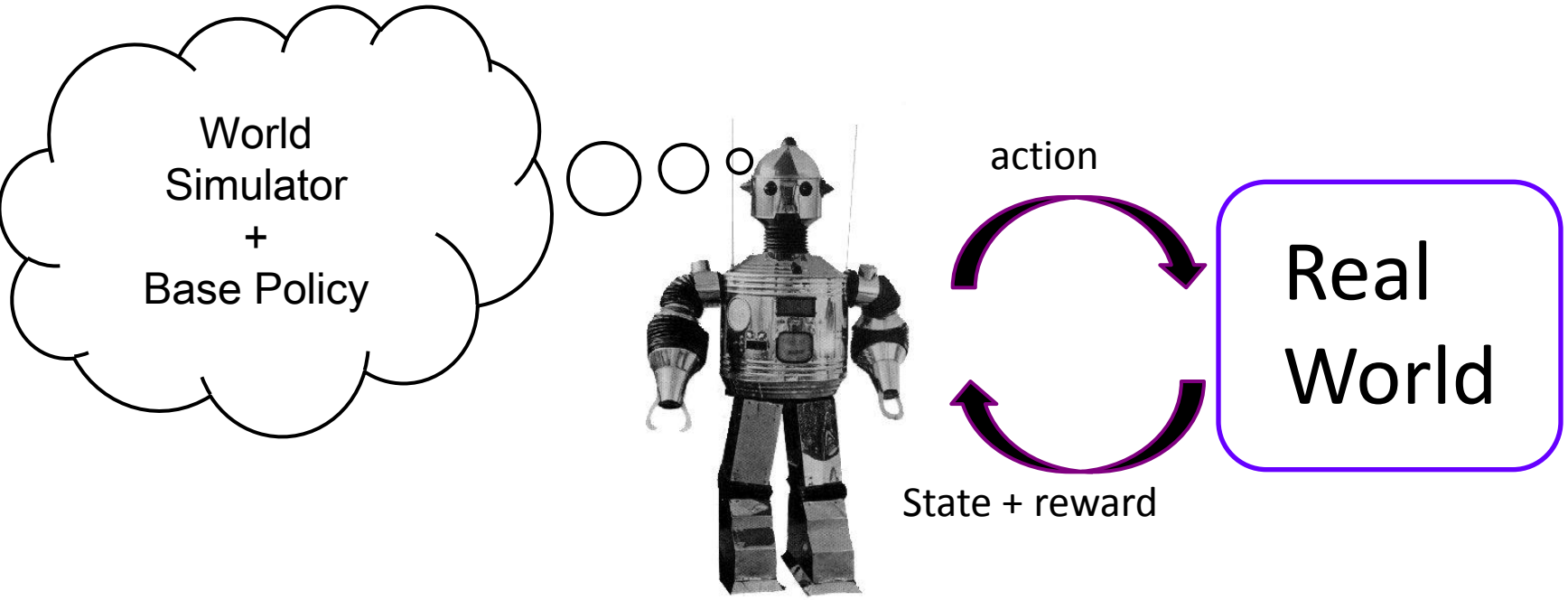
- Total simulator calls for PAC: $k \cdot w = O\left(\frac{k}{\varepsilon^2} \ln \frac{k}{\delta}\right)$
- Can get rid of ln(k) term with more complex algorithm [Even-Dar et. al., 2002].

Outline

- Uniform Sampling
  - PAC Bound for Single State MDPs
  - Policy Rollouts for full MDPs
- Adaptive Sampling
  - UCB for Single State MDPs
  - UCT for full MDPs

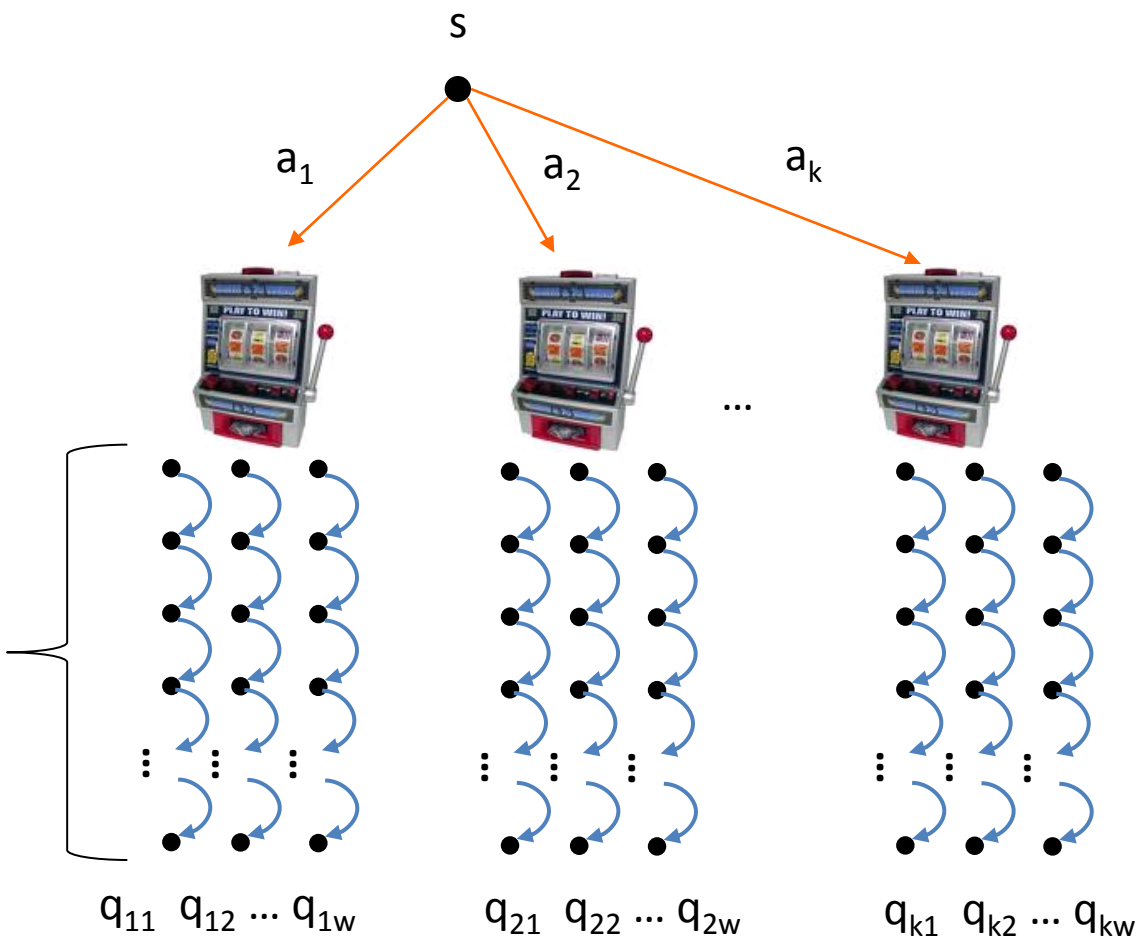
Policy Improvement via Monte-Carlo

- Now consider a multi-state MDP.
- Suppose we have a simulator and a non-optimal policy
  - E.g. policy could be a standard heuristic or based on intuition
- Can we somehow compute an improved policy?



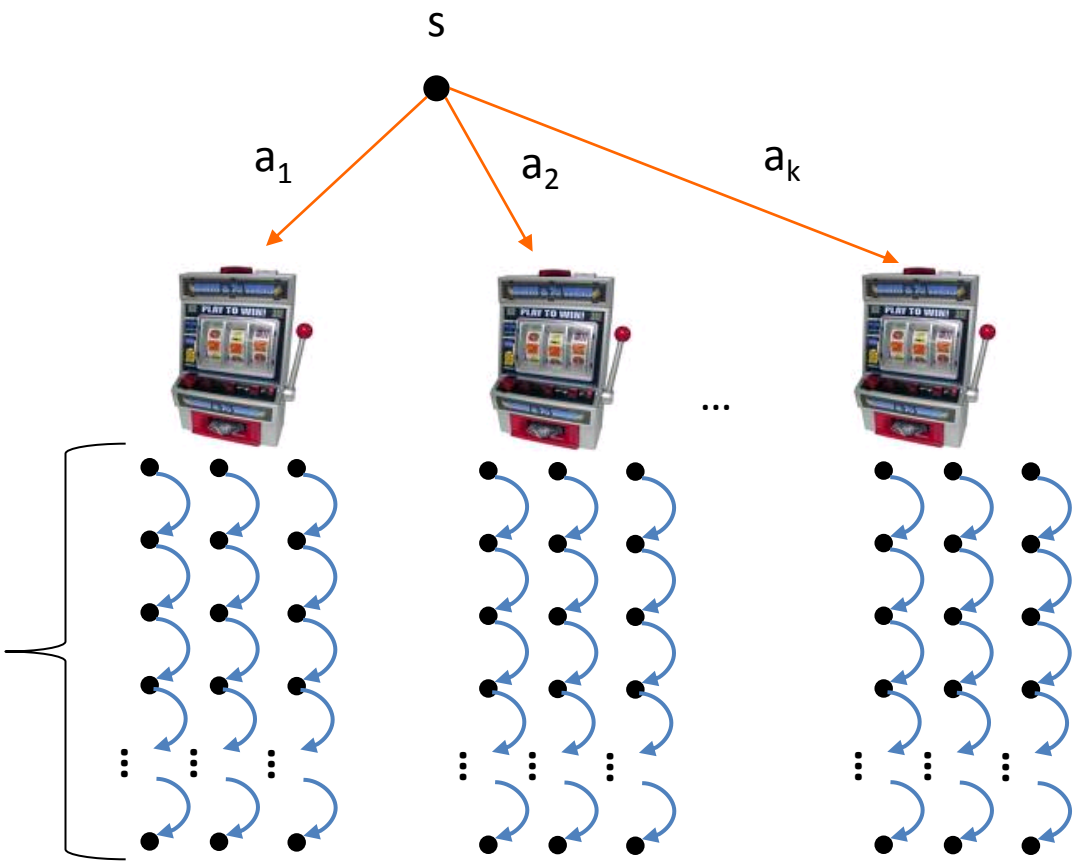
Policy Rollout Algorithm

1. For each a_i, run SimQ(s,a_i,π,h) **w** times
2. Return action with best average of SimQ results



Policy Rollout: # of Simulator Calls

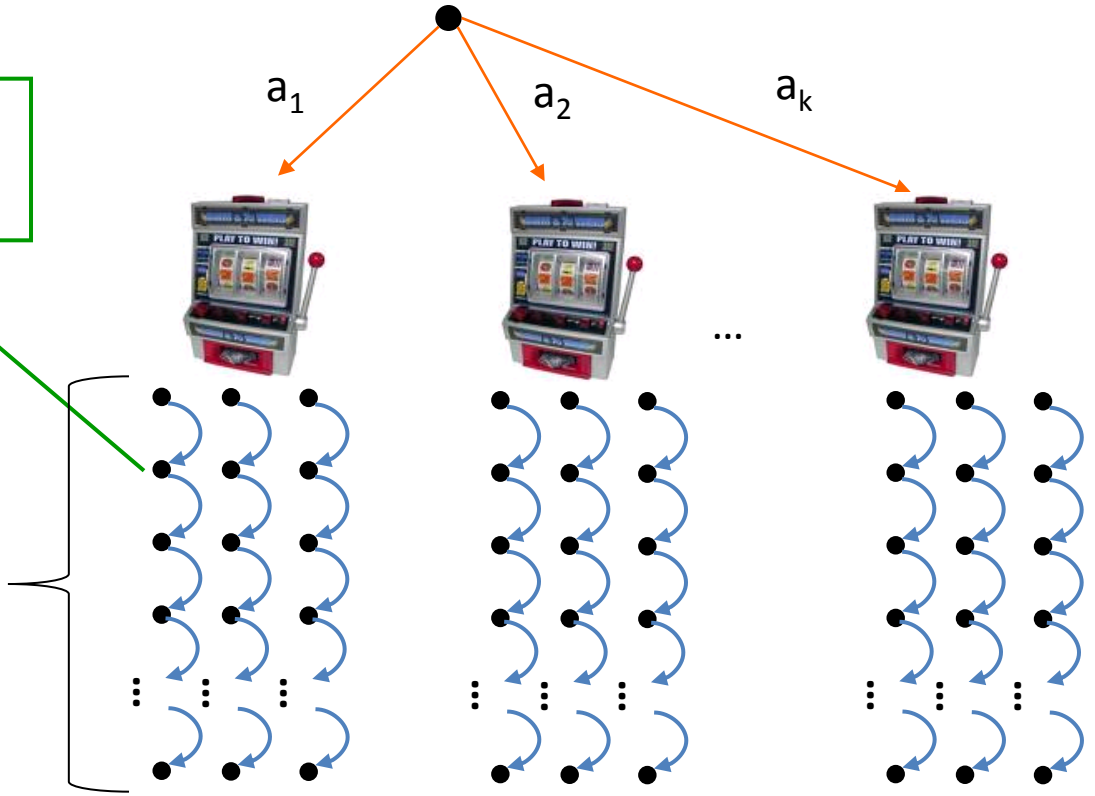
SimQ(s,a_i,π,h) trajectories
Each simulates taking action a_i then following π for h-1 steps.



- For each action, **w** calls to SimQ, each using h sim calls
- Total of khw calls to the simulator

Multi-Stage Rollout

Each step requires khw simulator calls



Trajectories of SimQ(s,a_i,Rollout(π),h)

- Two stage: compute **rollout policy** *of* **rollout policy** of π
- Requires $(khw)^2$ calls to the simulator for 2 stages
- In general exponential in the number of stages

Rollout Summary

- We often are able to write simple, mediocre policies
  - Network routing policy
  - Compiler instruction scheduling
  - Policy for card game of Hearts
  - Policy for game of Backgammon
  - Solitaire playing policy
  - Game of GO
  - Combinatorial optimization
- Policy rollout is a general and easy way to improve upon such policies
- Often observe substantial improvement!

Example: Rollout for Thoughtful Solitaire

[Yan et al. NIPS'04]

Player	Success Rate	Time/Game
Human Expert	36.6%	20 min
(naïve) Base Policy	13.05%	0.021 sec

Example: Rollout for Thoughtful Solitaire

[Yan et al. NIPS'04]

Player	Success Rate	Time/Game
Human Expert	36.6%	20 min
(naïve) Base Policy	13.05%	0.021 sec
1 rollout	31.20%	0.67 sec

Example: Rollout for Thoughtful Solitaire

[Yan et al. NIPS'04]

Player	Success Rate	Time/Game
Human Expert	36.6%	20 min
(naïve) Base Policy	13.05%	0.021 sec
1 rollout	31.20%	0.67 sec
2 rollout	47.6%	7.13 sec

Example: Rollout for Thoughtful Solitaire

[Yan et al. NIPS'04]

Player	Success Rate	Time/Game
Human Expert	36.6%	20 min
(naïve) Base Policy	13.05%	0.021 sec
1 rollout	31.20%	0.67 sec
2 rollout	47.6%	7.13 sec
3 rollout	56.83%	1.5 min
4 rollout	60.51%	18 min
5 rollout	70.20%	1 hour 45 min

Deeper rollout can pay off, but is expensive

Outline

- Uniform Sampling
  - PAC Bound for Single State MDPs
  - Policy Rollouts for full MDPs
- Adaptive Sampling
  - UCB for Single State MDPs
  - UCT for full MDPs

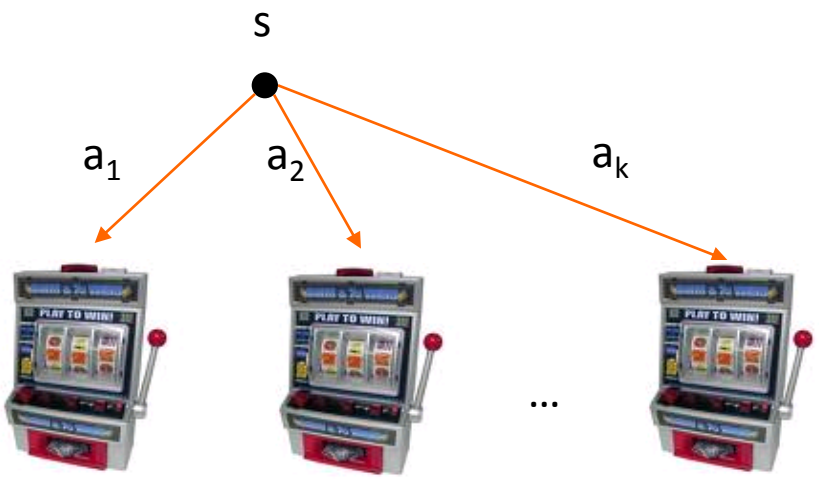
Non-Adaptive Monte-Carlo

- What is an issue with Uniform sampling?
  - time wasted equally on all actions!
  - no early learning about suboptimal actions
- Policy rollouts
  - Devotes equal resources to each state encountered in the tree
  - Would like to focus on most promising parts of tree

But how to control exploration of new parts of tree??

Regret Minimization Bandit Objective

- **Problem:** find arm-pulling strategy such that the expected total reward at time n is close to the best possible (i.e. pulling the best arm always)
  - UniformBandit is poor choice --- waste time on bad arms
  - Must balance **exploring** machines to find good payoffs and **exploiting** current knowledge



UCB Adaptive Bandit Algorithm

[Auer, Cesa-Bianchi, & Fischer, 2002]

- Q(a) : average payoff for action a based on current experience
- n(a) : number of pulls of arm a
- Action choice by UCB after n pulls:

Assumes payoffs in [0,1]

$$a^* = \arg\max_a Q(a) + \sqrt{\frac{2 \ln n}{n(a)}}$$

Value Term:
favors actions that looked good historically

Exploration Term:
actions get an exploration bonus that grows with ln(n)

Doesn't waste much time on sub-optimal arms unlike uniform!

UCB Algorithm [Auer, Cesa-Bianchi, & Fischer, 2002]

$$a^* = \arg\max_a Q(a) + \sqrt{\frac{2 \ln n}{n(a)}}$$

Theorem: expected number of pulls of sub-optimal arm **a** is bounded by:

$$\frac{8}{\Delta_a^2} \ln n$$

where Δ_a is regret of arm **a**

- Hence, the expected regret after *n* arm pulls compared to optimal behavior is bounded by O(log *n*)
- No algorithm can achieve a better loss rate

Outline

- Uniform Sampling
  - PAC Bound for Single State MDPs
  - Policy Rollouts for full MDPs
- Adaptive Sampling
  - UCB for Single State MDPs
  - UCT for full MDPs

UCB Based Policy Rollout

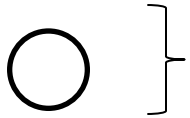
- Allocate samples non-uniformly
  - based on UCB action selection
  - More sample efficient than uniform policy rollout
  - Still suboptimal.

UCT Algorithm [Kocsis & Szepesvari, 2006]

- Instance of Monte-Carlo Tree Search
  - Applies principle of UCB
  - Some nice theoretical properties
  - Better than policy rollouts – asymptotically optimal
  - Major advance in computer Go
- Monte-Carlo Tree Search
  - Repeated Monte Carlo simulation of a rollout policy
  - Each rollout adds one or more nodes to search tree
- Rollout policy depends on nodes already in tree

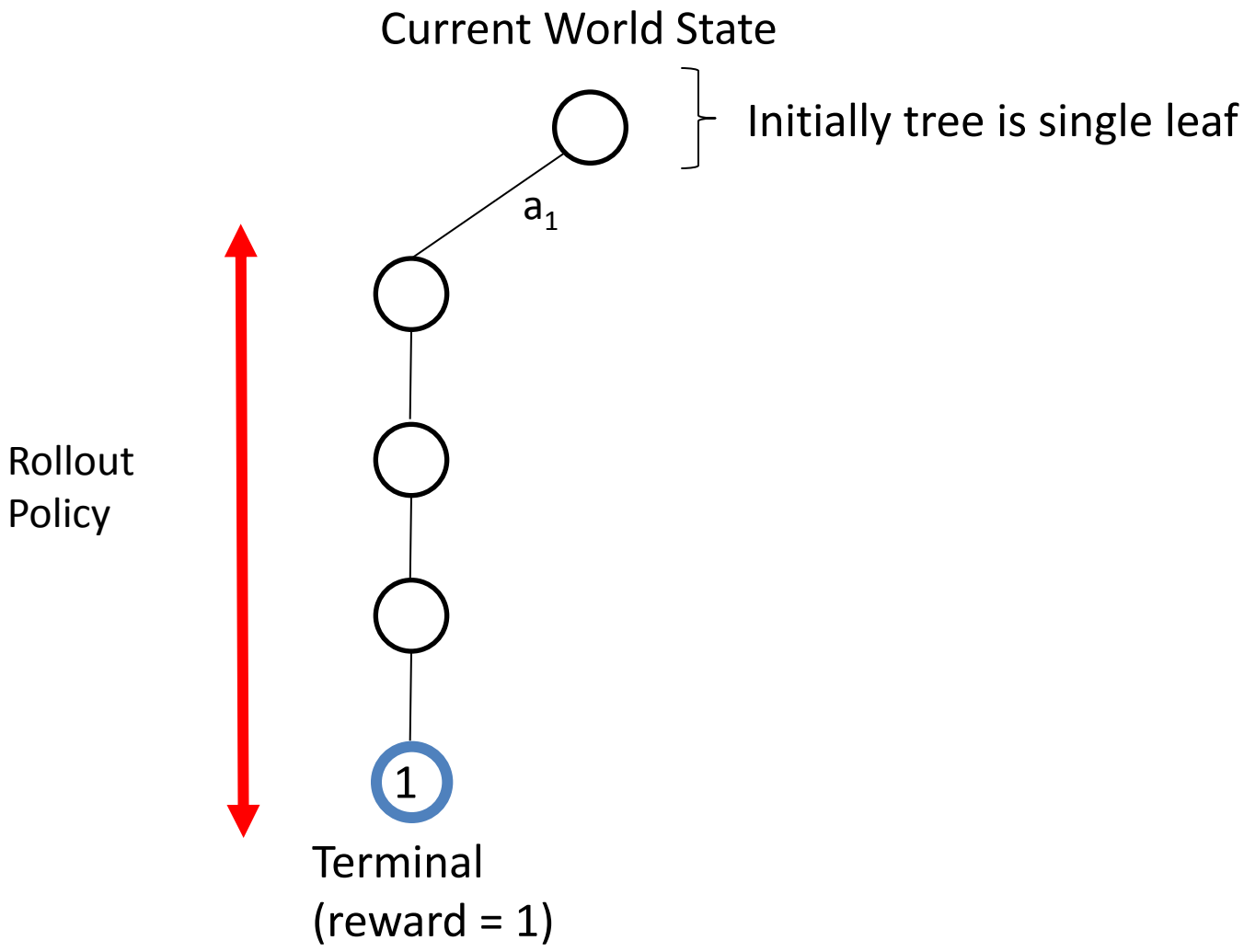
At a leaf node perform a random rollout

Current World State

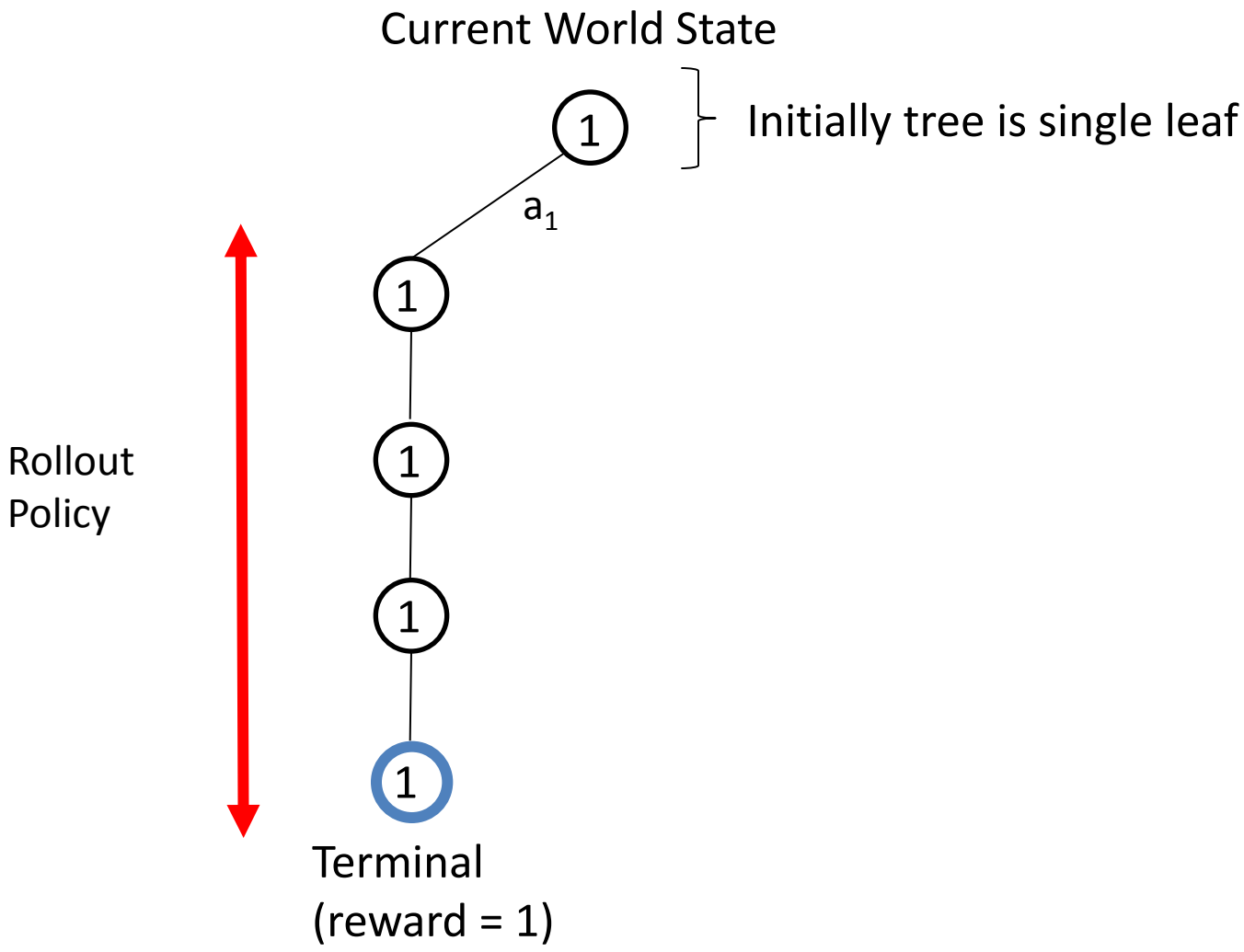


Initially tree is single leaf

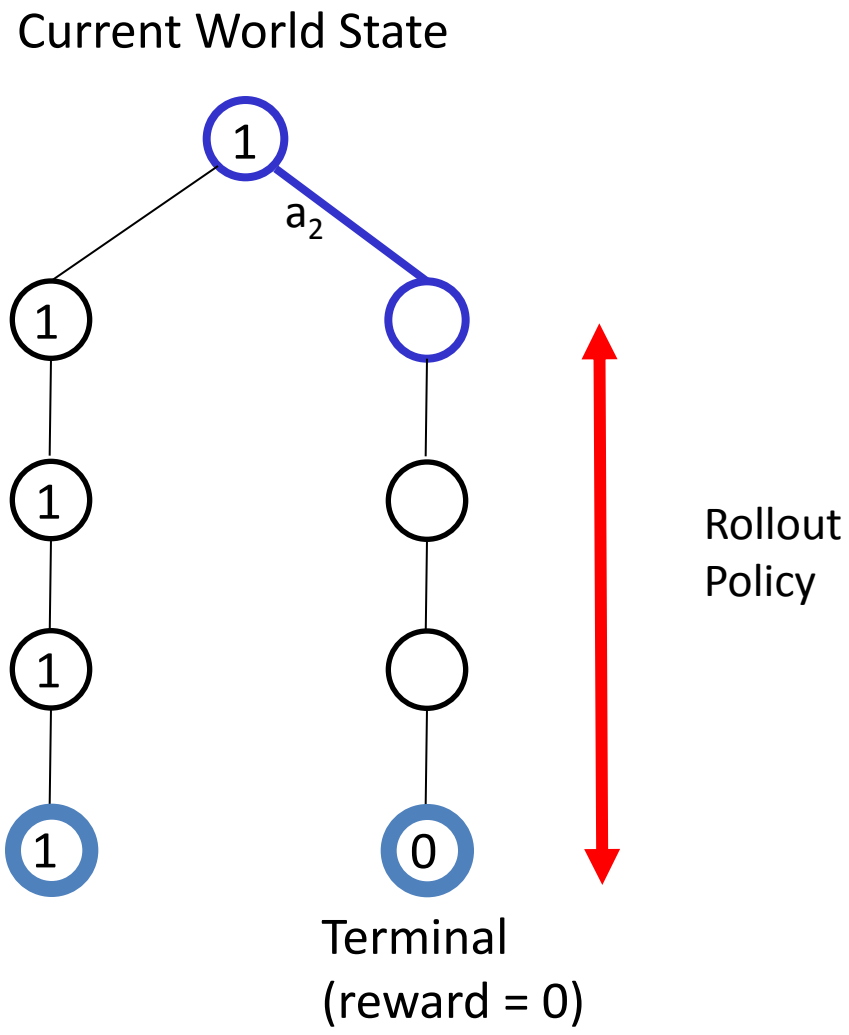
At a leaf node perform a random rollout



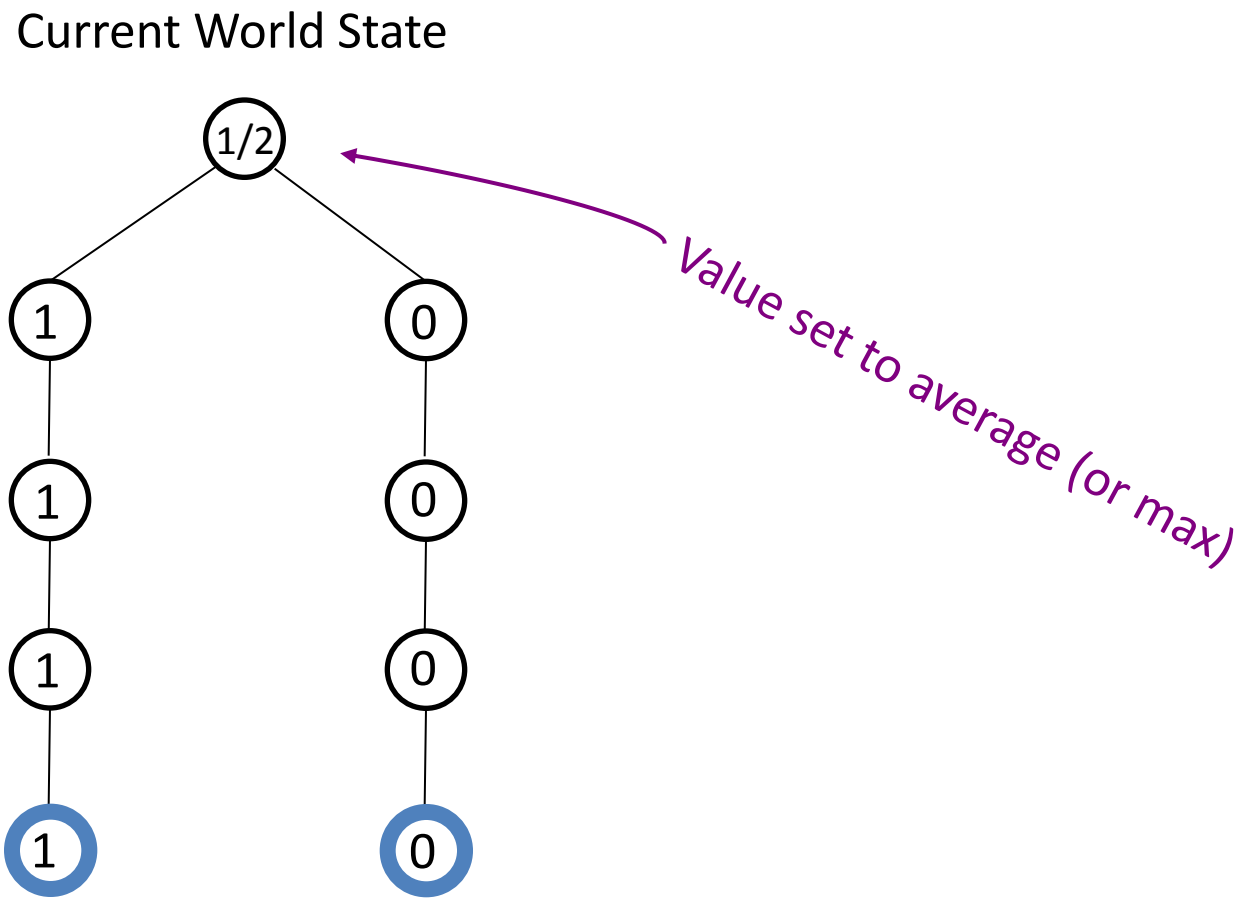
At a leaf node perform a random rollout



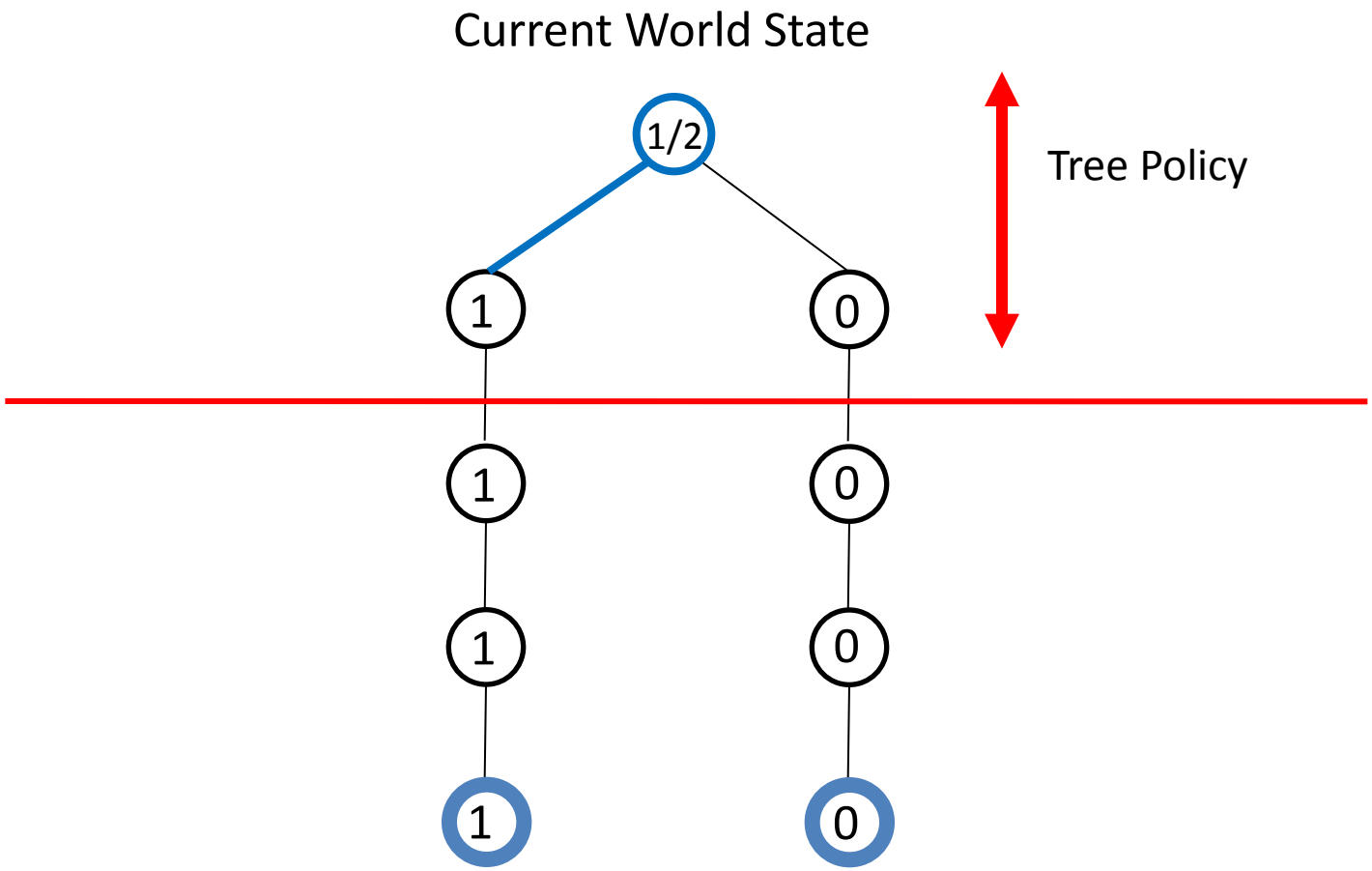
Must select each action at a node at least once



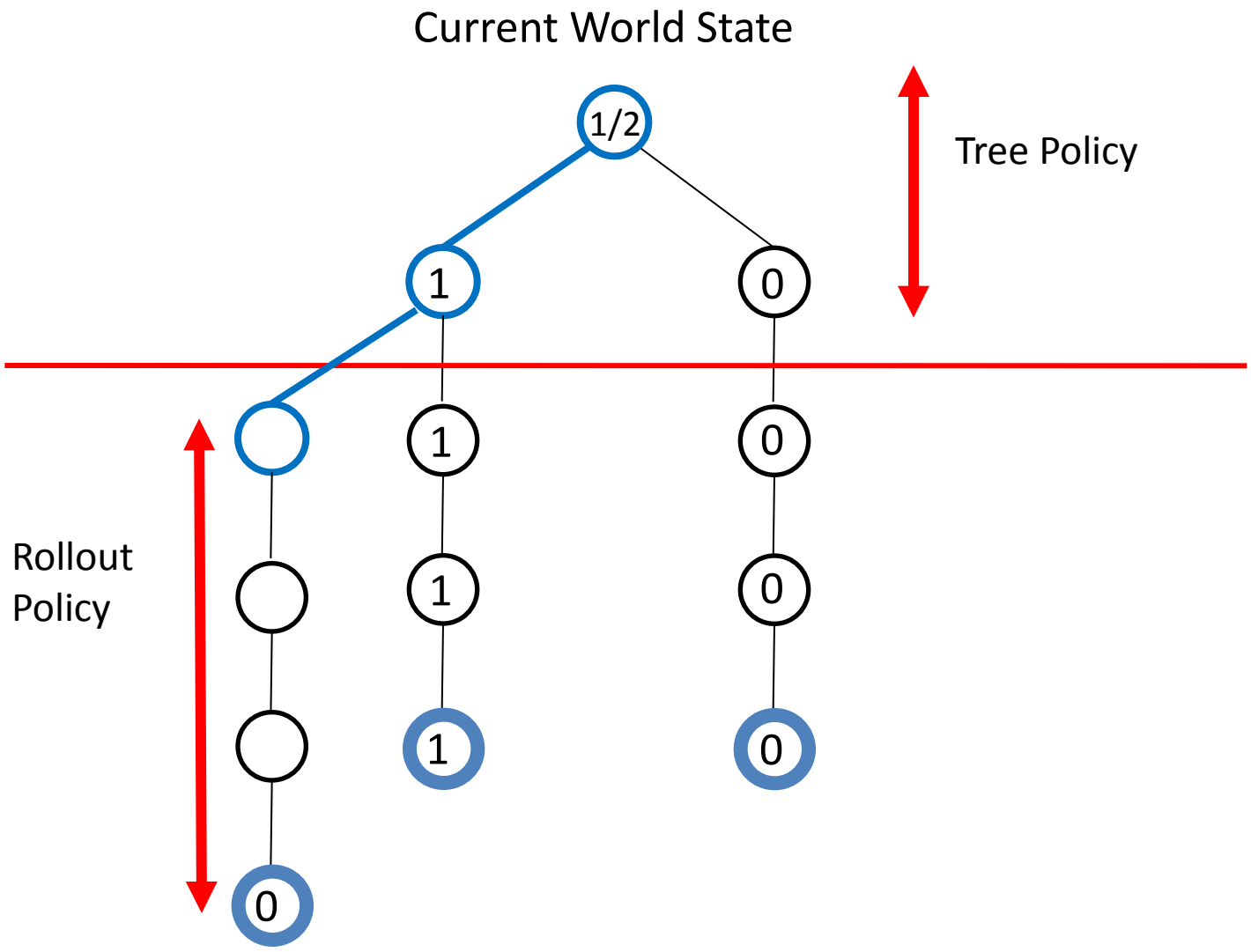
Must select each action at a node at least once



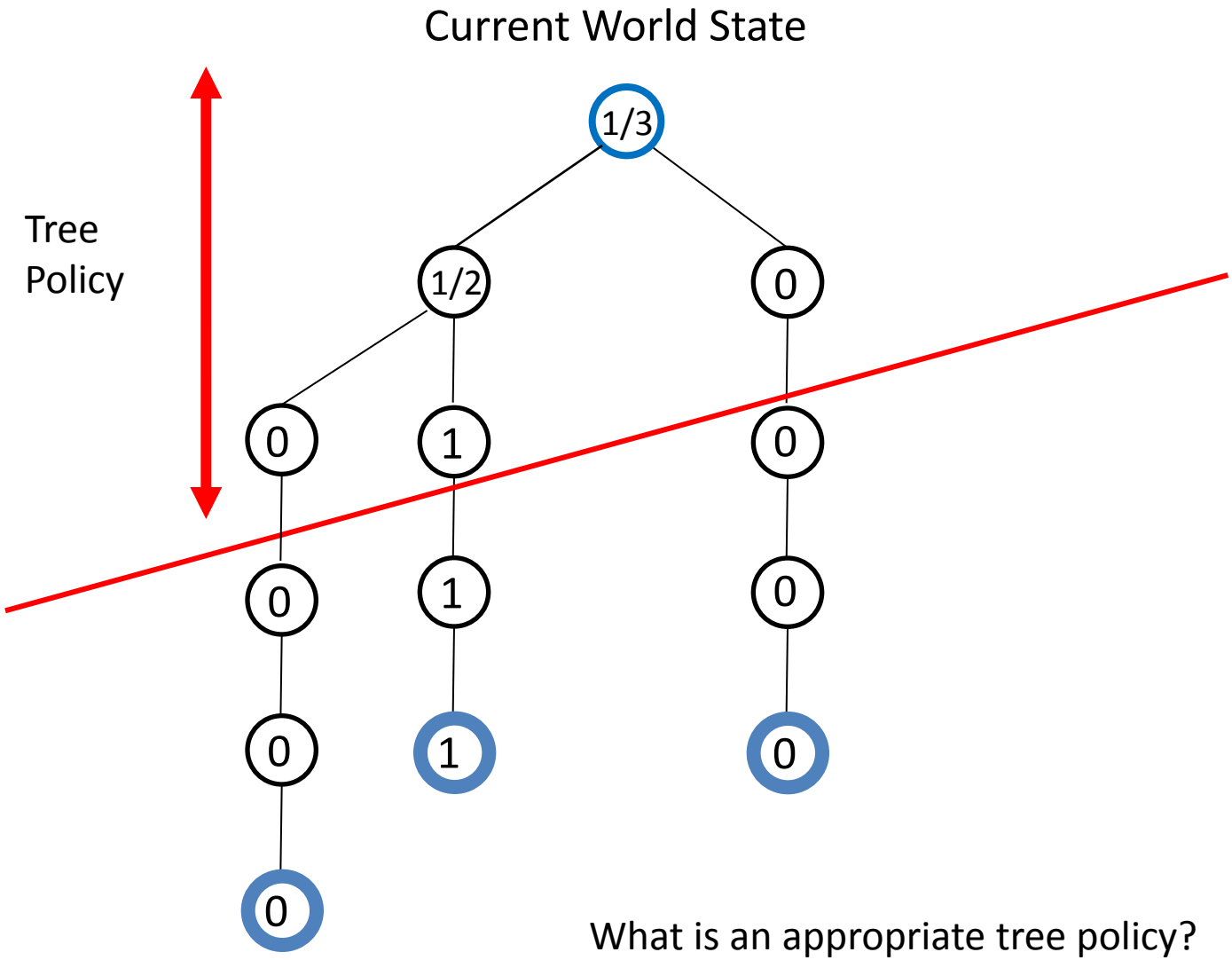
When all node actions tried once, select action according to tree policy



When all node actions tried once, select action according to tree policy



When all node actions tried once, select action according to tree policy



What is an appropriate tree policy?
Rollout policy?

UCT Algorithm [Kocsis & Szepesvari, 2006]

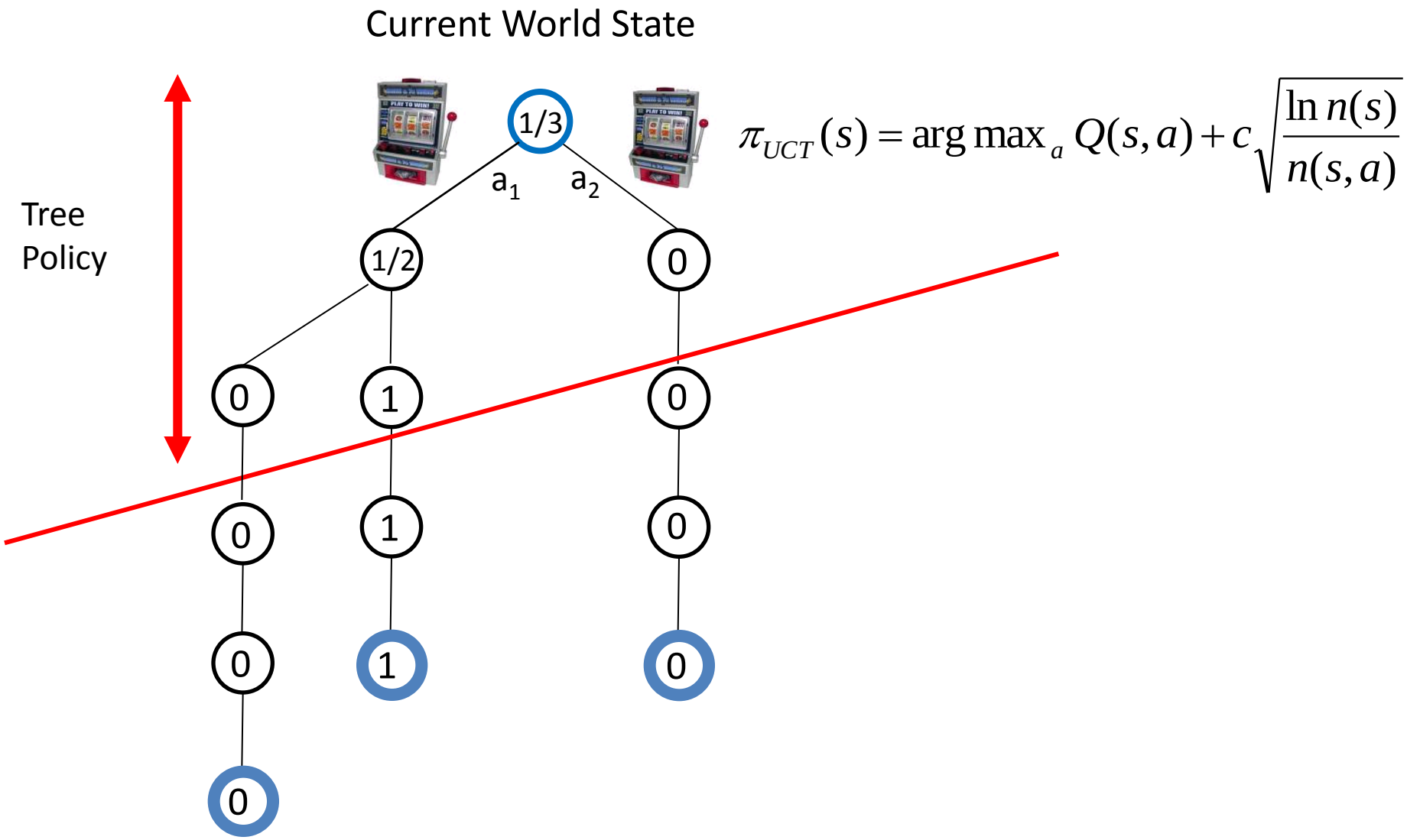
- Basic UCT uses random rollout policy

- Tree policy is based on UCB:
  - Q(s,a) : average reward received in current trajectories after taking action a in state s
  - n(s,a) : number of times action a taken in s
  - n(s) : number of times state s encountered

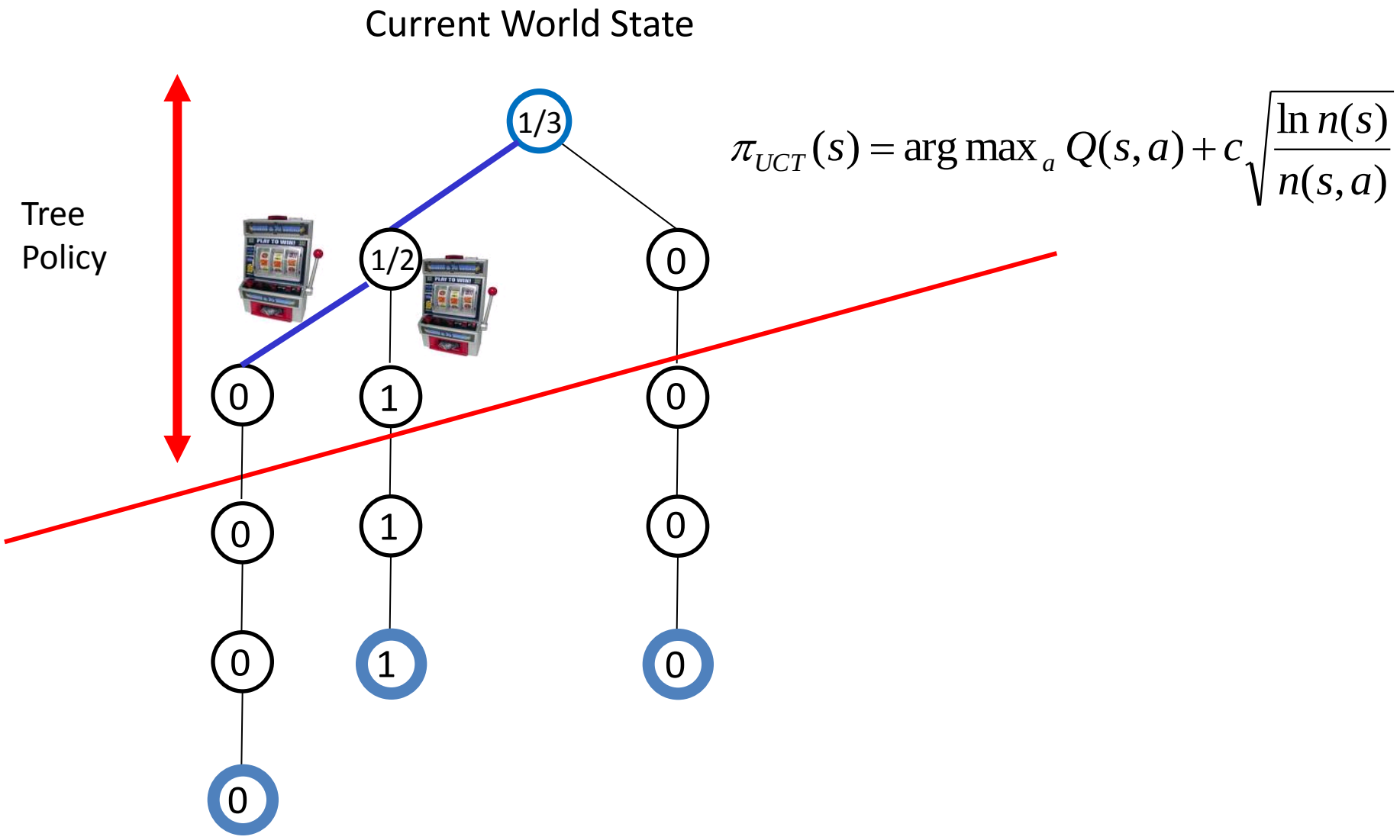
$$\pi_{UCT}(s) = \arg\max_a Q(s,a) + c\sqrt{\frac{\ln n(s)}{n(s,a)}}$$

Theoretical constant that must be selected empirically in practice

When all node actions tried once, select action according to tree policy



When all node actions tried once, select action according to tree policy

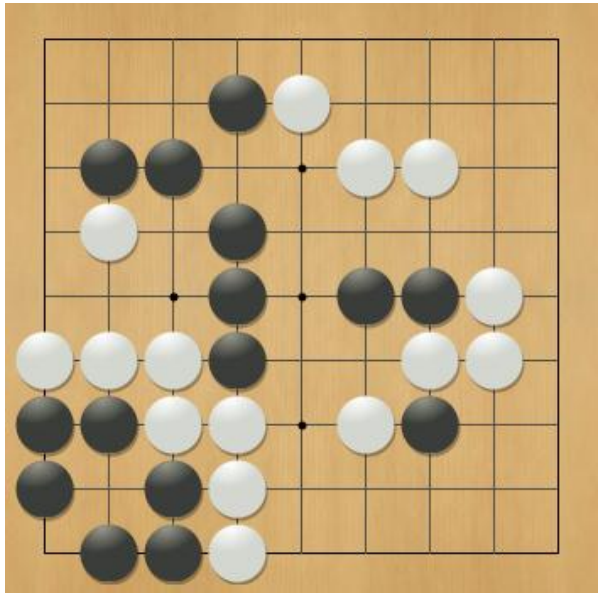


UCT Recap

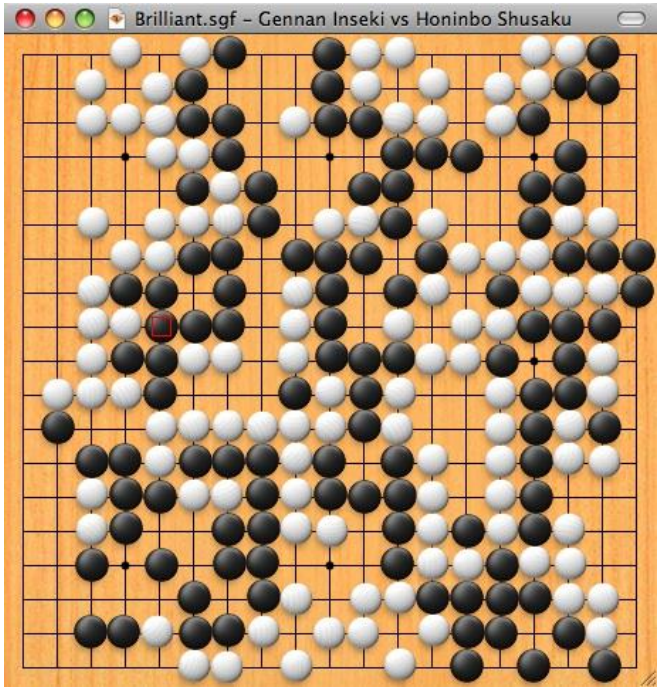
- To select an action at a state s
  - Build a tree using N iterations of monte-carlo tree search
    - Default policy is uniform random
    - Tree policy is based on UCB rule
  - Select action that maximizes Q(s,a)
    (note that this final action selection does not take the exploration term into account, just the Q-value estimate)

- The more simulations the more accurate

Computer Go



9x9 (smallest board)



19x19 (largest board)

- “Task Par Excellence for AI” (Hans Berliner)
- “New Drosophila of AI” (John McCarthy)
- “Grand Challenge Task” (David Mechner)

Game of Go

human champions refuse to compete against computers, because software is too bad.

	Chess	Go
Size of board	8 x 8	19 x 19
Average no. of moves per game	100	300
Avg branching factor per turn	35	235
Additional complexity		Players can pass

A Brief History of Computer Go

- *2005*: Computer Go is impossible!
- *2006*: UCT invented and applied to 9x9 Go *(Kocsis, Szepesvari; Gelly et al.)*
- *2007*: Human master level achieved at 9x9 Go *(Gelly, Silver; Coulom)*
- *2008*: Human grandmaster level achieved at 9x9 Go *(Teytaud et al.)*

- *ELO rating 1800 → 2600*

Other Successes

- Klondike Solitaire (wins 40% of games)
- General Game Playing Competition
- Real-Time Strategy Games
- Combinatorial Optimization

- Probabilistic Planning (MDPs)

- Usually extend UCT is some ways

Improvements/Issues

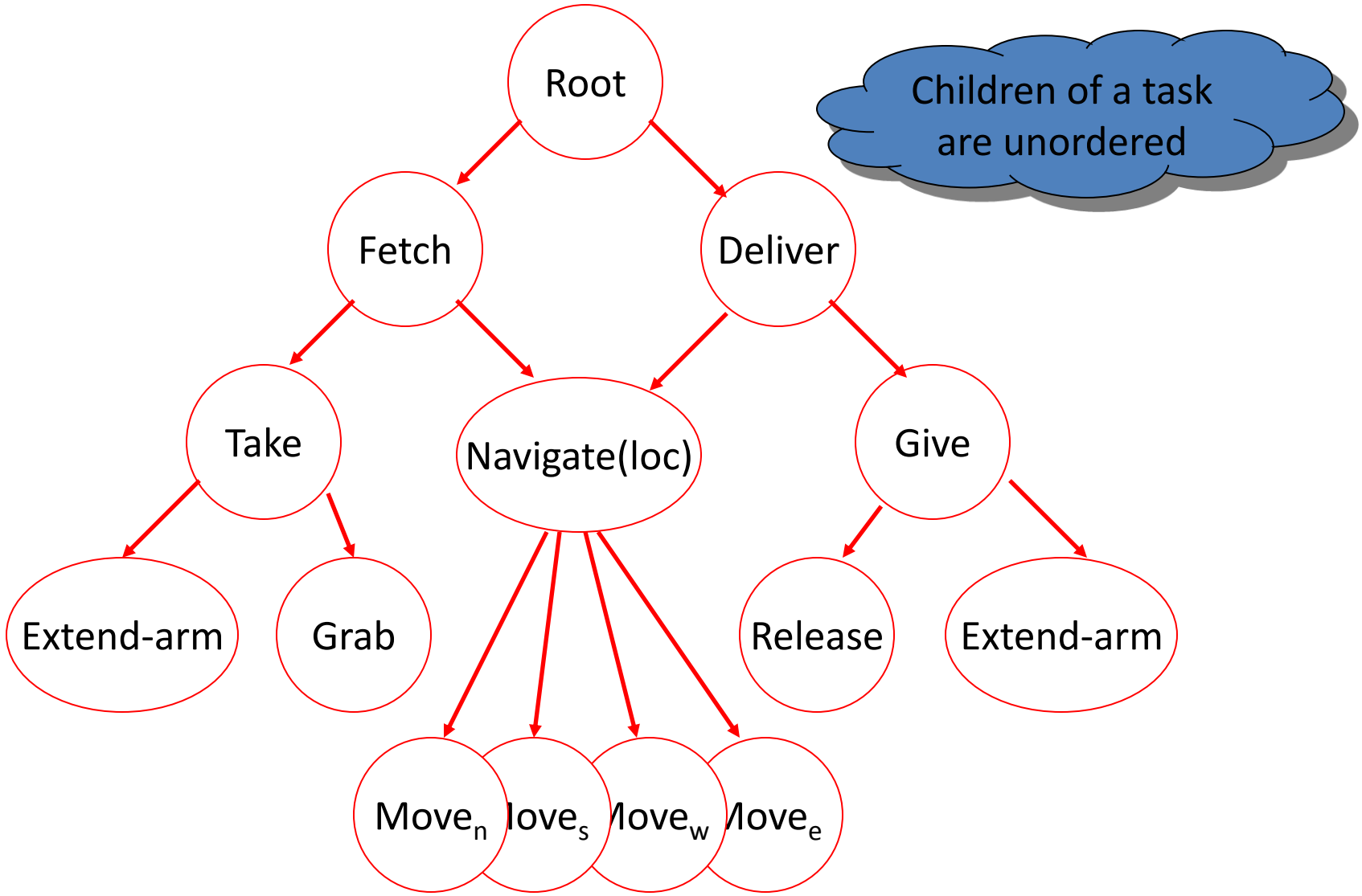
- Use domain knowledge to improve the base policies
  - E.g.: don't choose obvious stupid actions
  - better policy does not imply better UCT performance
- Learn a heuristic function to evaluate positions
  - Use heuristic to initialize leaves
- *Interesting question: UCT versus minimax*

Generalization in Learning

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to even hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Task Hierarchy: MAXQ Decomposition [Dietterich'00]



Summary

- Multi-armed Bandits
  - Principles of both RL and Monte-Carlo
- Reinforcement Learning
  - Exploration/Exploitation tradeoff
  - Passive/Active RL
  - Model free/Model based
- Monte-Carlo Planning
  - Exploration/Exploitation tradeoff
  - Uniform/Adaptive Sampling