Value Iteration and Monte Carlo

Tutorials PUI 2017/2018

Jan Mrkos

MDP

- Method for solving MDPs iteratively
- Value function of a policy:

$$V_\pi(s) = \mathbb{E}[\sum_{t=0} \gamma^t \cdot R(s_t, a_t, s_{t+1}) \,|s_0 = s, a_t = \pi(s_t)]$$

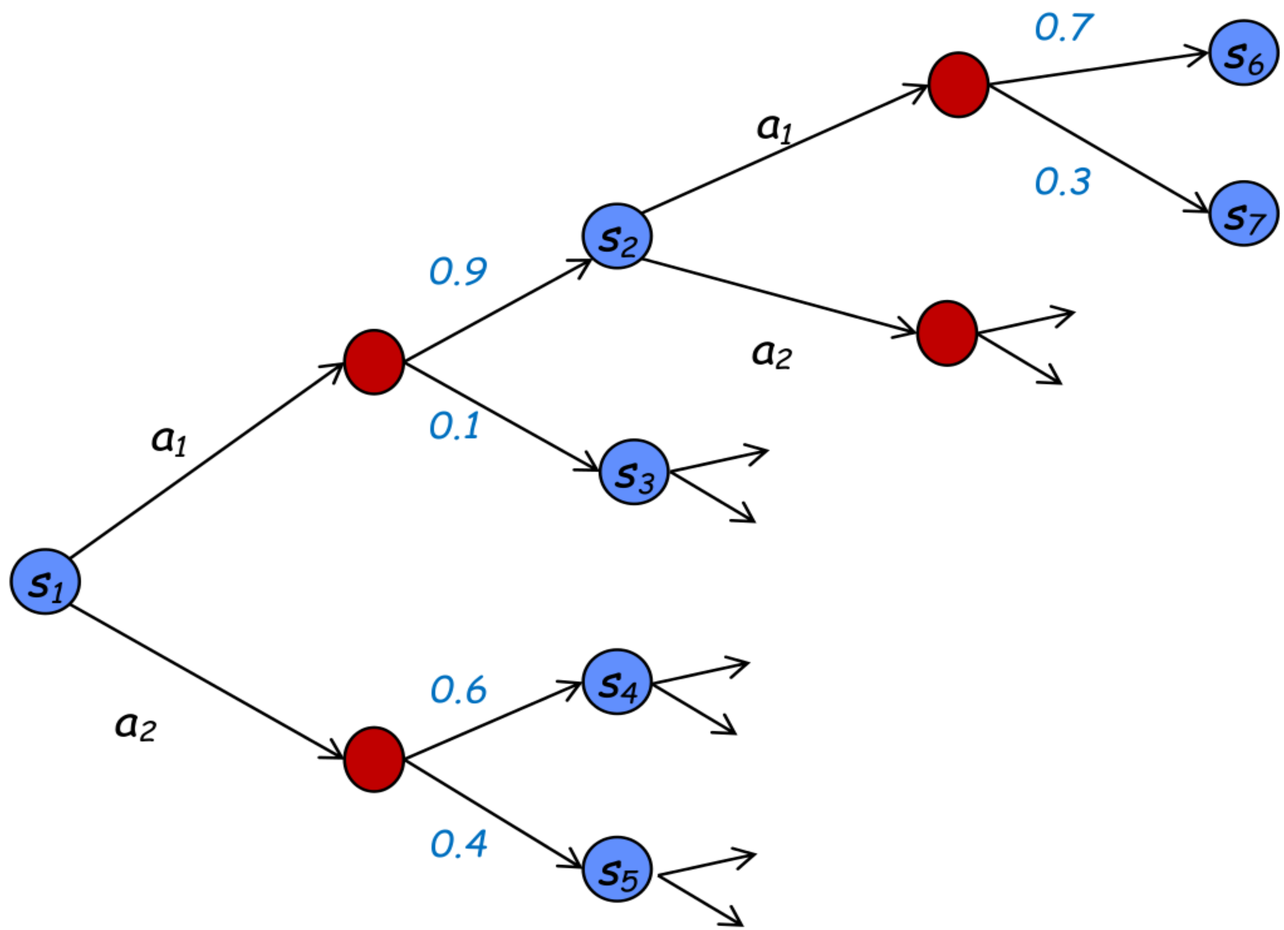
from state s_0, policy π, reward function R

- Optimal value function:

$$V^*(s) = \max_\pi \boldsymbol{E}[R(s, \pi)]$$

- Optimal policy – gives max value in each state

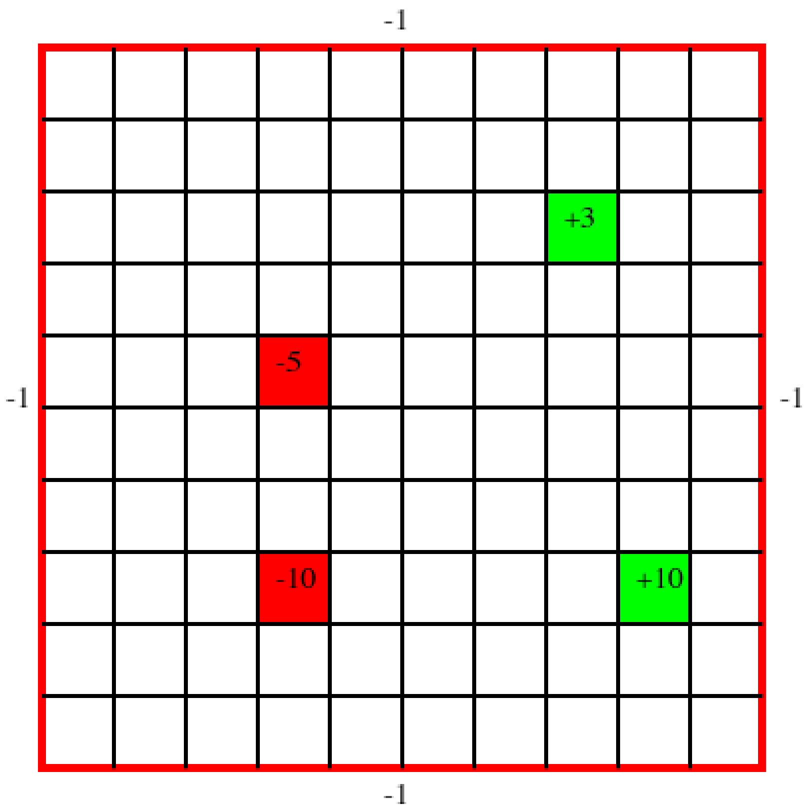
MDP

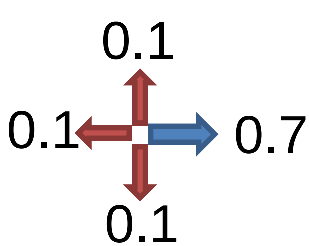


Value Iteration

- basic algorithm for solving MDPs based on Bellman's equation
- **Value iteration (Bellman backup)**
  - $V^0(s) = 0 \quad \forall s \in S$
  - $V'(s) = \max_{a \in A} \sum_{s' \in S} \underbrace{T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]}_{\text{Q-function } (Q(s,a))}$
  - for $k \to \infty$ values converge to optimum $V^k \to V^*$

Value Iteration – example





Step: 0

0	0	0
0	0	0
0	0	0

Step: 1

0	0	-0.1
0	10	-0.1
0	0	-0.1

Step: 2

0	6.3	-0.1
6.3	9.8	6.2
0	6.3	-0.1

Step: 3

4.5	6.2	4.4
6.2	9.7	6.6
4.5	6.1	4.4

From http://artint.info/html/ArtInt_224.html#gridworld-ex

Value Iteration - questions

- Why $V^0(s) = 0$?
- "for $k \rightarrow \infty$ values converge to optimum $V^k \rightarrow V^*$"
  - ∞??? Yeah, that sounds useful
  - How fast is this?

- Value iteration stopping criterion (Bellman error):

$$\left\|V - V'\right\| < \epsilon$$

Gives ϵ-optimal value function V

Relation between policy and value function

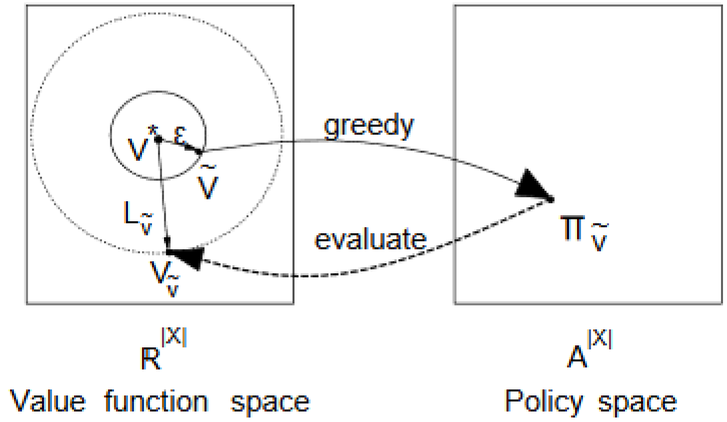
- Distance between value functions:

$$||V - V'|| = ||V - V'||_{\infty} = \max_{s} |V(s) - V'(s)|$$

Theorem:

The value function V^{π} of greedy policy π derived from ϵ-optimal value function V satisfies following:

$$||V - V^{*}|| < 2\epsilon\gamma/(1-\gamma)$$



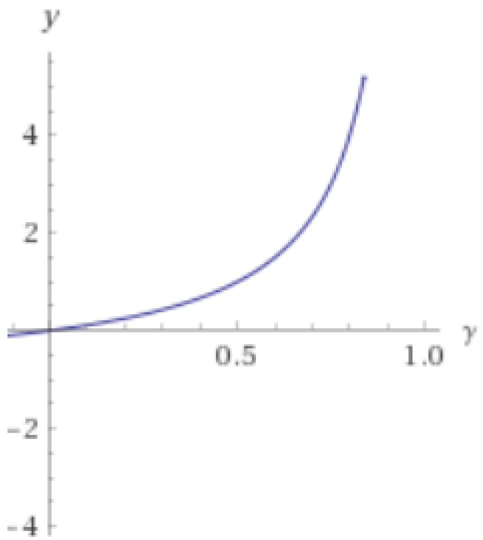


Image from Singh and Yee, An Upper Bound on the Loss from Approximate Optimal-Value Functions

Proof

Theorem:

The value function V^{π} of greedy policy π derived from ϵ-optimal value function V satisfies following:

$$\left||V - V^*|\right| < 2\epsilon\gamma/(1-\gamma)$$

Proof:

Value iteration - options

- Can keep V or Q in memory
  - Saving V(s) arrays – less storage
  - Saving Q(s, a) – less iterations
- Asynchronous value iteration
  - Keep only one array of value functions, update online
  - Less space and faster convergence
  - Difficulty with the stopping condition

Order of Backups

- Heuristic search VI
  - Uses heuristics to dynamically determine the order of updates (lecture)
  - Even static ordering beforehand can have huge impact
- Prioritized sweeping
  - Updates states that will produce largest delta in value function
- Topological value iteration
  - Builds a graph of "casually dependent" states to find optimal order of backups.

Best Action Only

- In VI, all actions have to be backed up
- Policy iteration – actions according to one policy only
- Requires "optimistic estimate" of Q^*: $Q(x, a) > Q^*(x, a)$

Monte Carlo

- Non-Adaptive Monte-Carlo
  - **Single state case (PAC Bandit)**
  - Policy rollouts
- Adaptive Monte-Carlo
  - Single state case (UCB Bandit)
  - UCT MCTS

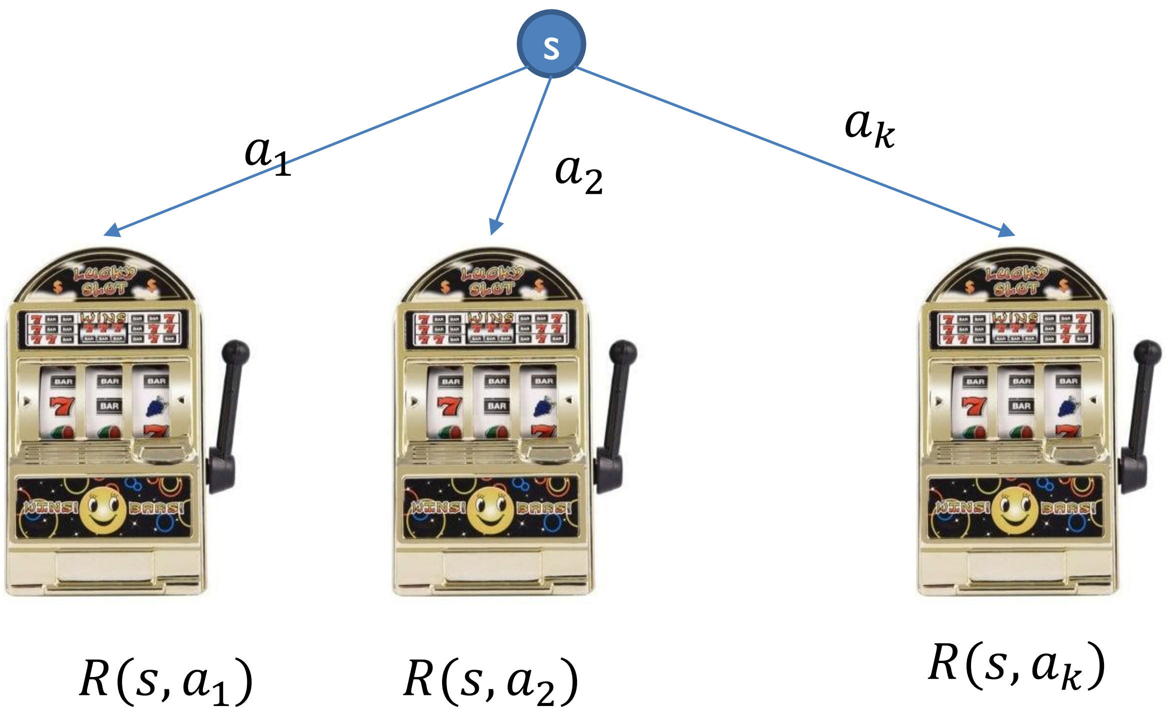
Following slides used from from Alan Fern, Dan Weld, Dan Klein, Luke Zettlmoyer

Monte-Carlo and MDPs

- Exact state space description not available in large state spaces, but there exist simulators:
  - Traffic simulations
  - Robotics simulators
  - Go
- Monte-Carlo in MDPs
  - Use simulator to evaluate stochastically selected actions
  - Finite (but large) state set S
  - Finite action set A
  - Stochastic, real-valued, bounded reward function R(s, a)=r
  - Stochastic transition function T(s,a)=s'

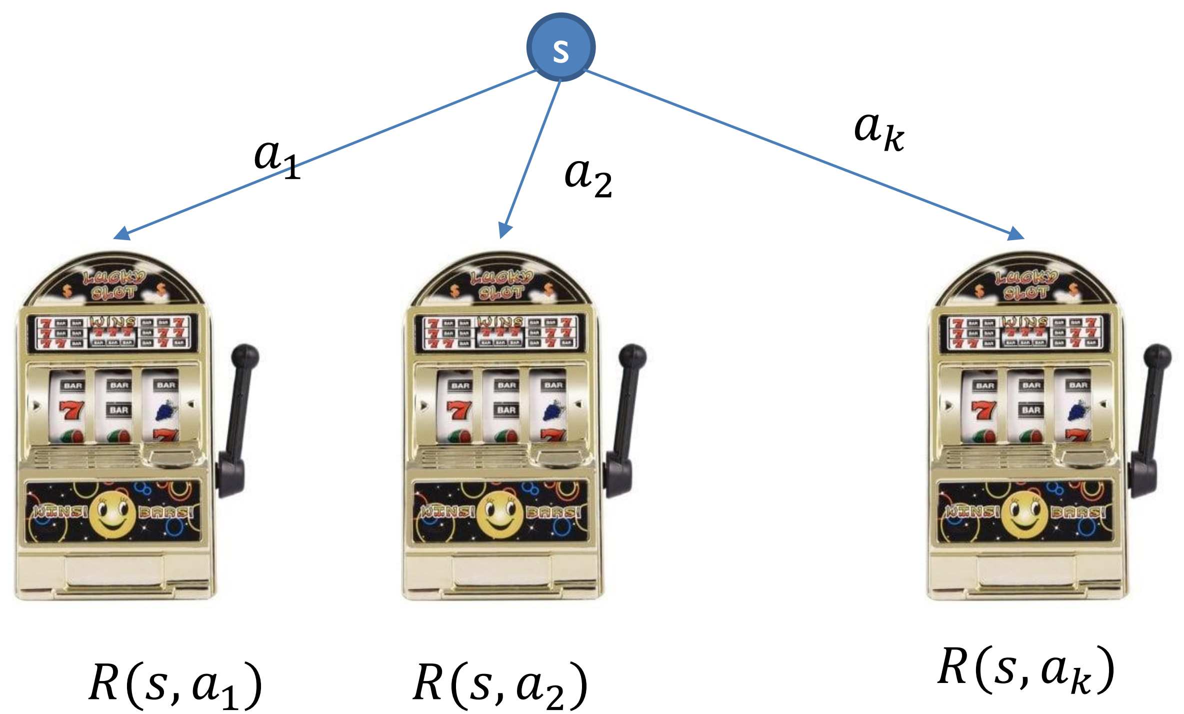
Planning in single state

- Multi-Armed Bandit Problem
  - Which action will yield best expected reward?
  - Simulator returns reward R(s, a)



Multi-Armed bandit - PAC objective

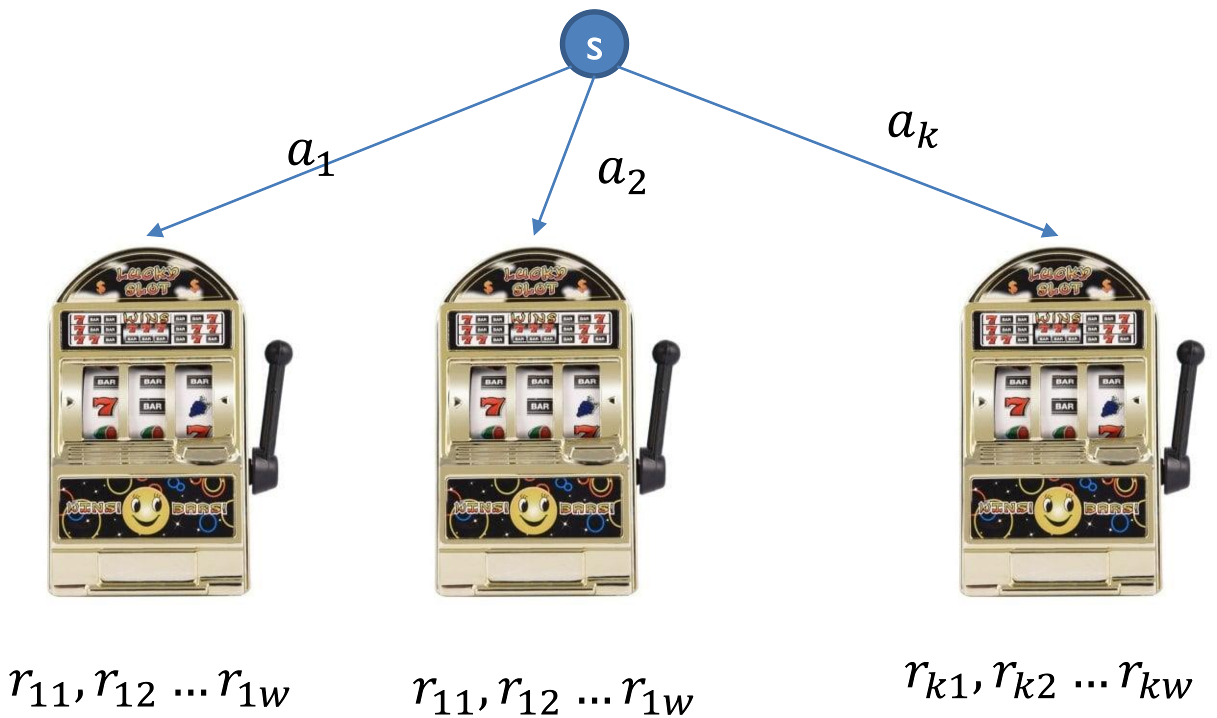
- PAC = Probably Approximately Correct
- Select arm that **probably** (prob. $1-\delta$) has **approximately** (within ϵ) the best expected reward.
- Use least possible number of runs.



Uniform Bandit Algorithm

1. Pull each arm **w** times
2. Return arm with best average reward

Q: How many times do we have to pull?



Naïve bandit from Even-Dar et. al., 2002

Uniform Bandit PAC bound

- Markov's (and then Chebyshev's) inequality:

Random variable $X \geq 0$ and $c > 0$. Then for any real c,

$$P(X \geq c) < \frac{E(X)}{c}$$

Random variable X with finite E(X) and variance $\sigma^2 > 0$.
Then for any $c > 0$,

$$P(|X - E(X)| \geq k\sigma) \leq \frac{1}{k^2}$$

- Markov's inequality gives Chernoff Bound that can be used to calculate the probability of within close to some value

PAC Objective and Bound

- Select arm that **probably** (prob. $1-\delta$) has **approximately** (within ϵ) the best expected reward.

If $w \geq \left(\frac{R_{max}}{\epsilon}\right)^2 \ln\frac{k}{\delta}$ then for all arms with probability $1-\delta$

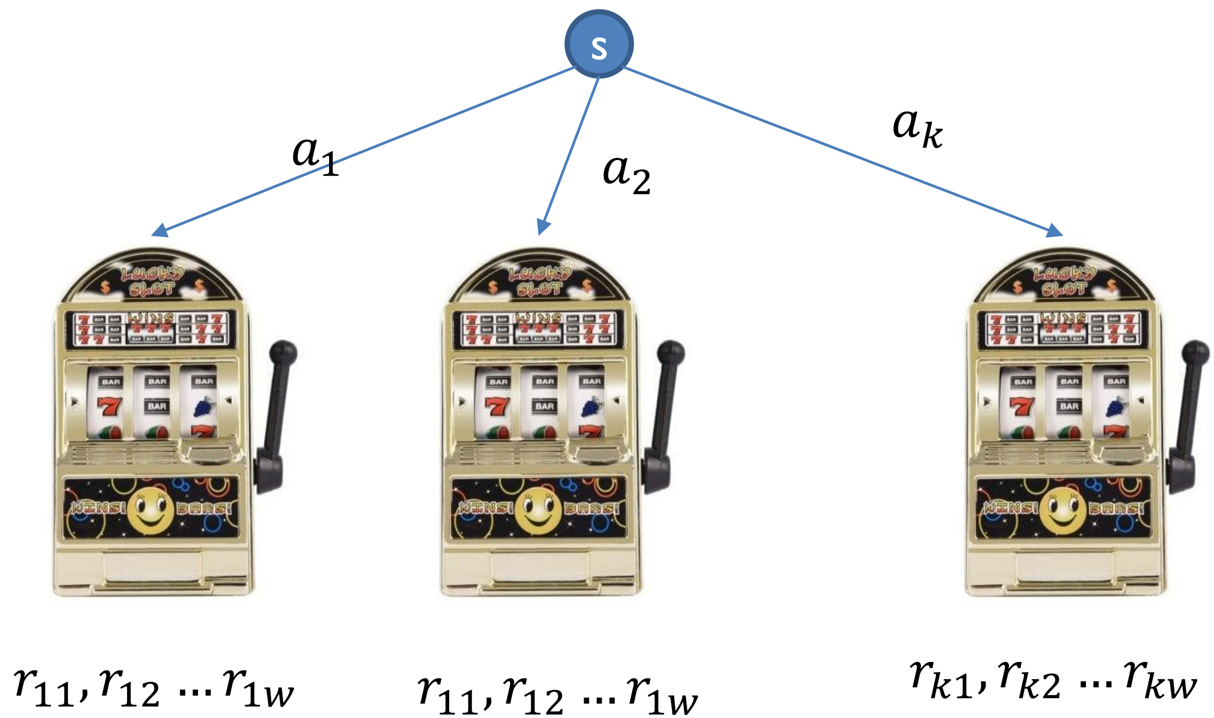
$$\left| E[R(s,a_i)] - \frac{1}{w}\sum_{j=1}^{w} r_{ij} \right| \leq \epsilon$$

- Meaning all action estimates are ϵ accurate with probability $1-\delta$, R_{max} is maximal reward.

Uniform Bandit Algorithm

For PAC, we need this many calls:

$$k\ w \geq O\left(\frac{\mathrm{k}}{\epsilon^2}\ln\frac{k}{\delta}\right)^2$$



Naïve bandit from Even-Dar et. al., 2002

Monte Carlo

- Non-Adaptive Monte-Carlo
  - Single state case (PAC Bandit)
  - **Policy rollouts**
- Adaptive Monte-Carlo
  - Single state case (UCB Bandit)
  - UCT MCTS

Slides from Alan Fern, Dan Weld, Dan Klein, Luke Zettlmoyer

Policy Improvement using Monte-Carlo

- Assume non-optimal policy and simulator
- How can you improve the policy?

Policy Improvement Theorem

- Q-function $Q_\pi(s,a)$ is defined as:

$$Q_\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t,a_t)\,|s_0 = s, a_t = \pi(s_t)\right]$$

  That is, expected total discounted reward of starting in s, taking action a and then following policy π.

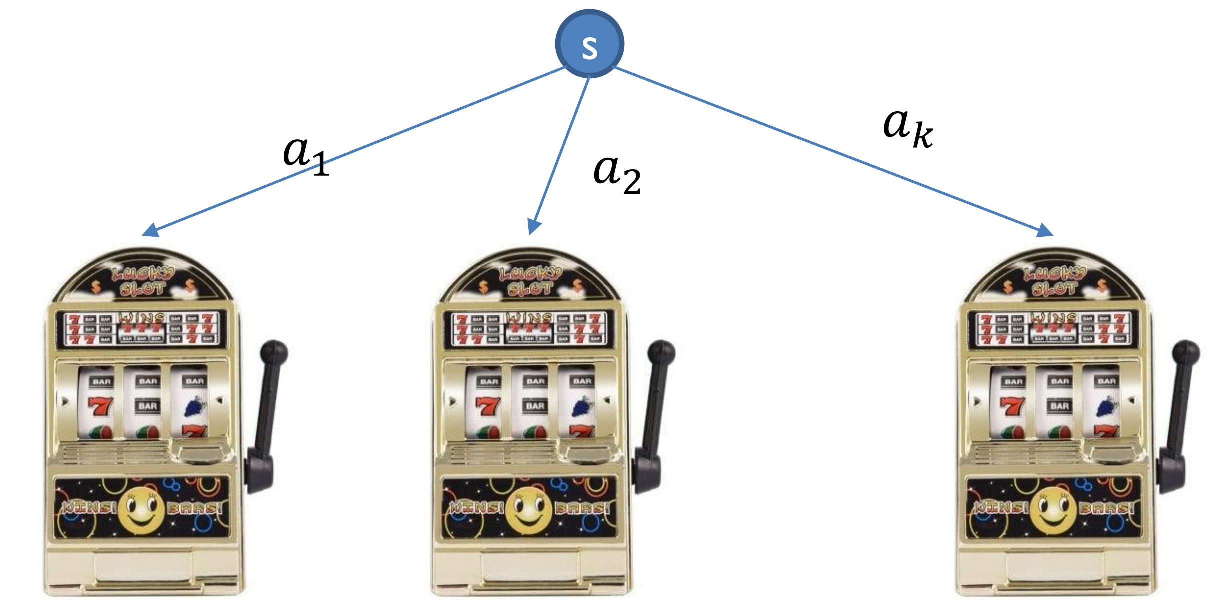
- Let $\pi'(s) = argmax_a Q_\pi(s,a)$
- Theorem (Howard, 1960)

For any non-optimal policy π the policy π' is a strict improvement over π.

- Computing π' amounts to finding action that maximizes Q-function of π (similar to policy iteration).
  - How do we apply the bandit idea?

Policy Improvement via Bandits

- **Idea:** define stochastic function $SimQ(s, a, \pi)$ that we can implement and that will have expected value $Q_\pi(s, a)$
- Next, just use bandit algorithm to determine best action
- **How to implement SimQ?**



$SimQ(s, a_1, \pi)$ $SimQ(s, a_2, \pi)$ $SimQ(s, a_k, \pi)$

Q-value Estimation

- SimQ might be implemented by simulating the execution of action a in state s and then following π thereafter.
  - For infinite horizon, this would never stop!
  - We approximate using finite horizon
- For horizon h, Q-function $Q_\pi(s,a,h)$ is defined as:

$$Q_\pi(s,a,h) = \mathbb{E}\left[\sum_{t=0}^{h-1} \gamma^t R(s_t,a_t)\,|s_0 = s, a_t = \pi(s_t)\right]$$

That is, expected total discounted reward of starting in s, taking action a and then following policy π for h-1 steps.

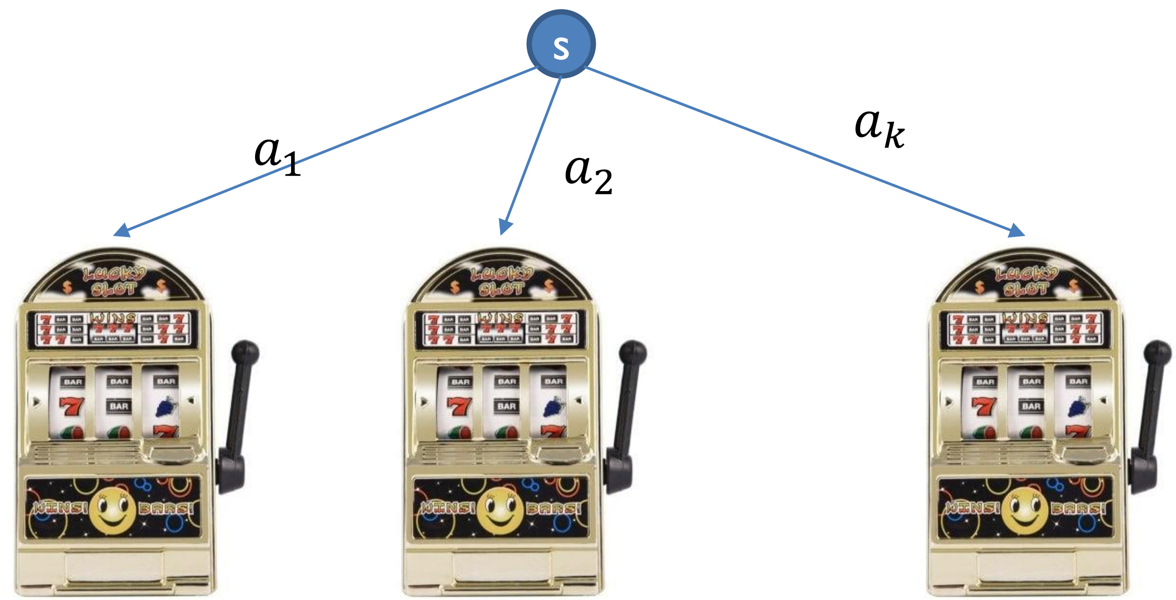
- What is the approximation error? Exponential in h:

$$|Q_\pi(s,a) - Q_\pi(s,a,h)| \le \gamma^h V_{max} \qquad V_{max} = \frac{R_{max}}{1-\gamma}$$

(prove in class for 1 point (5 minutes))

Policy Improvement via Bandits

- **Better idea:** redefine stochastic function $SimQ(s, a, \pi, h)$ that we can implement and that will have expected value $Q_\pi(s, a, h)$
- Next, just use bandit algorithm to determine best action
- **How to implement SimQ?**



$SimQ(s, a_1, \pi)$ $SimQ(s, a_2, \pi)$ $SimQ(s, a_k, \pi)$

Policy improvement via Bandits

SimQ(s,a,π,h)

```
r=R(s,a)              Simulate a in s
s=T(s,a)
for i=1 to h-1
      r=r+γ^i R(s,π(s))     Simulate h-1 steps
      s=T(s, π(s))          of policy
return r
```

- Implementat exaxtly as the formula suggests. Simulate taking action a in s and follow policy π for h steps. Return discounted sum of rewards.
- Expected value of SimQ(s,a,π,h) is SimQ(s,a,π), which can be made arbitrarily close to $Q_{\pi}(s,a)$ by increasing h. (**why?**)

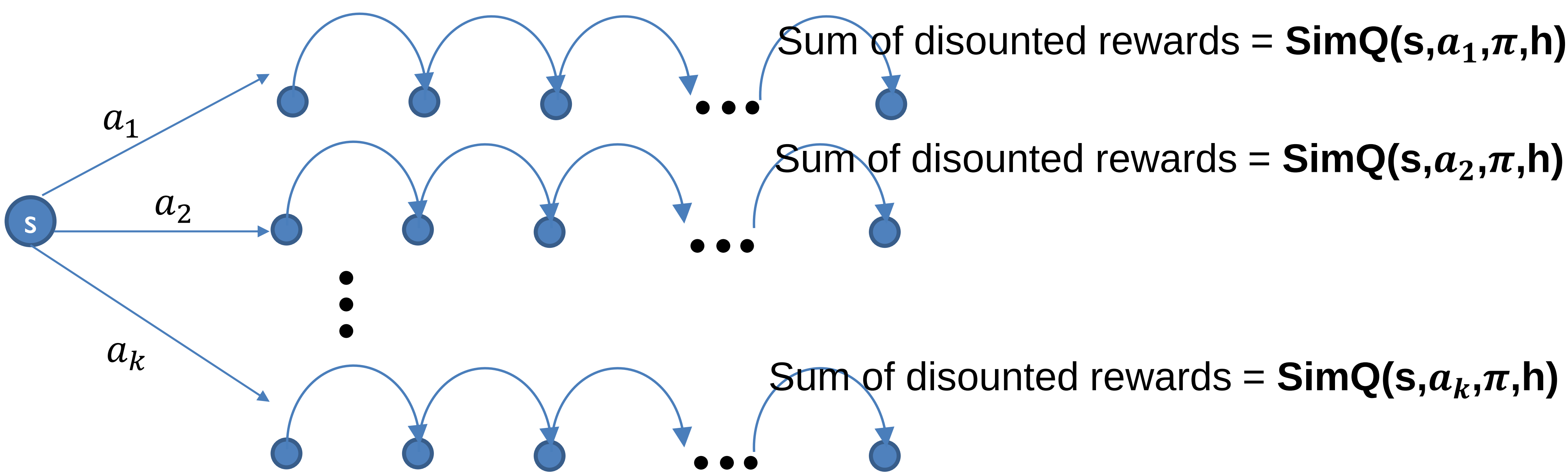
Policy improvement via Bandits

SimQ(s,a,π,h)

r=R(s,a)
s=T(s,a)
Simulate a in s

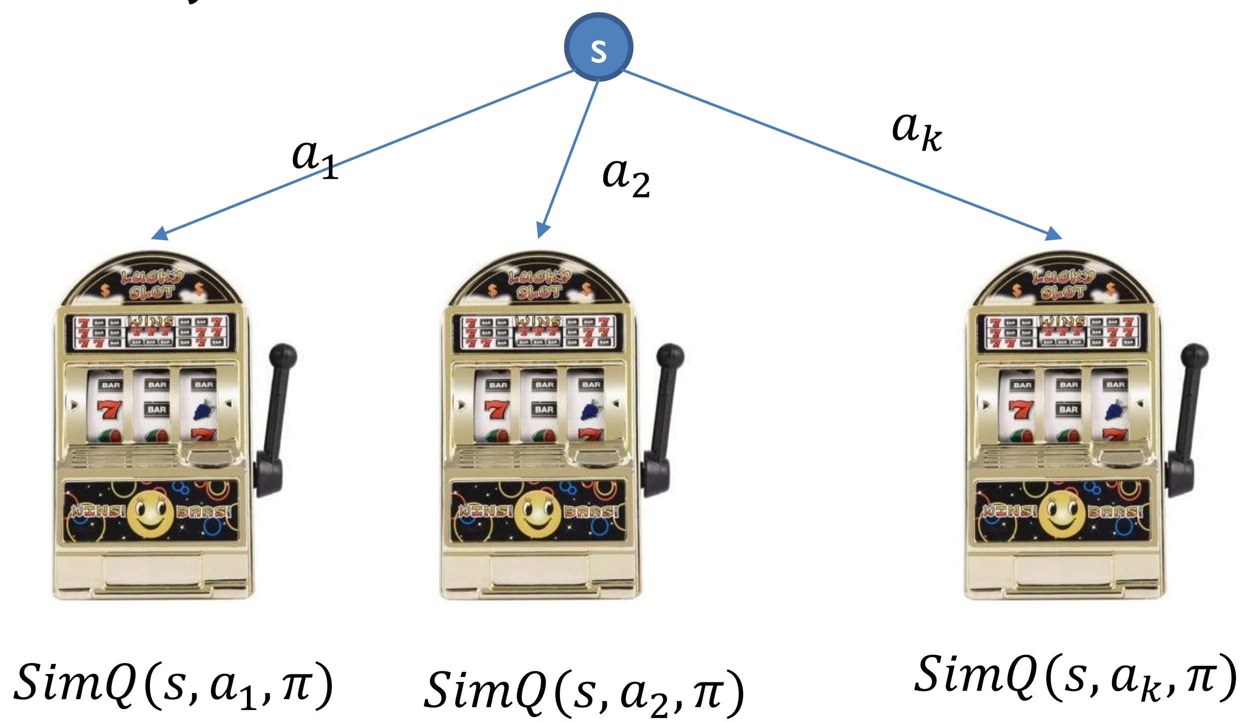
for i=1 to h-1
  r=r+γ^iR(s,π(s))
  s=T(s, π(s))
Simulate h-1 steps of policy

return r



Policy Improvement via Bandits

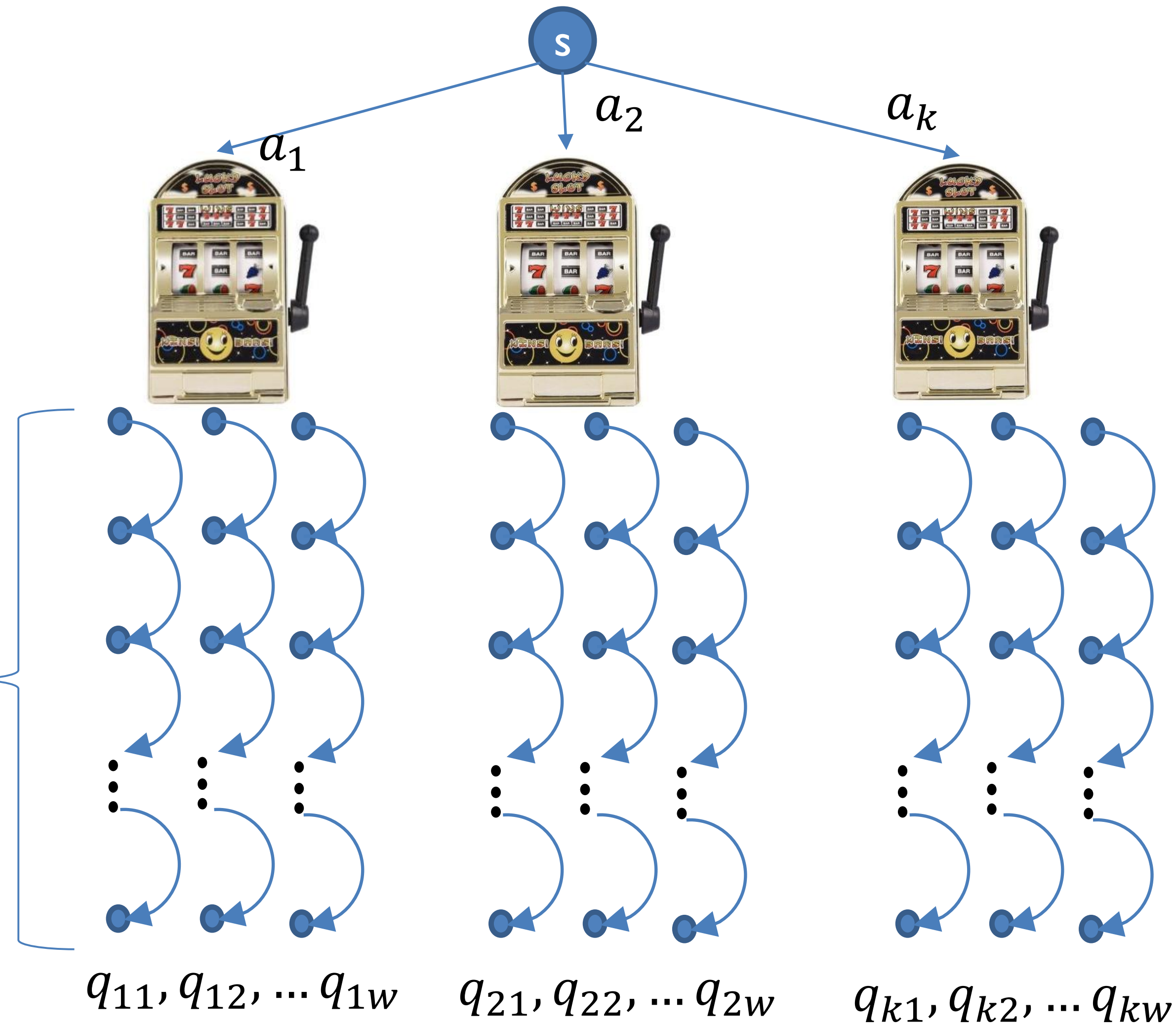
- **Better idea:** redefine stochastic function $SimQ(s, a, \pi, h)$ that we can implement and that will have expected value $Q_\pi(s, a, h)$
- Next, just use bandit algorithm to determine best action
- **Apply the PAC Objective**



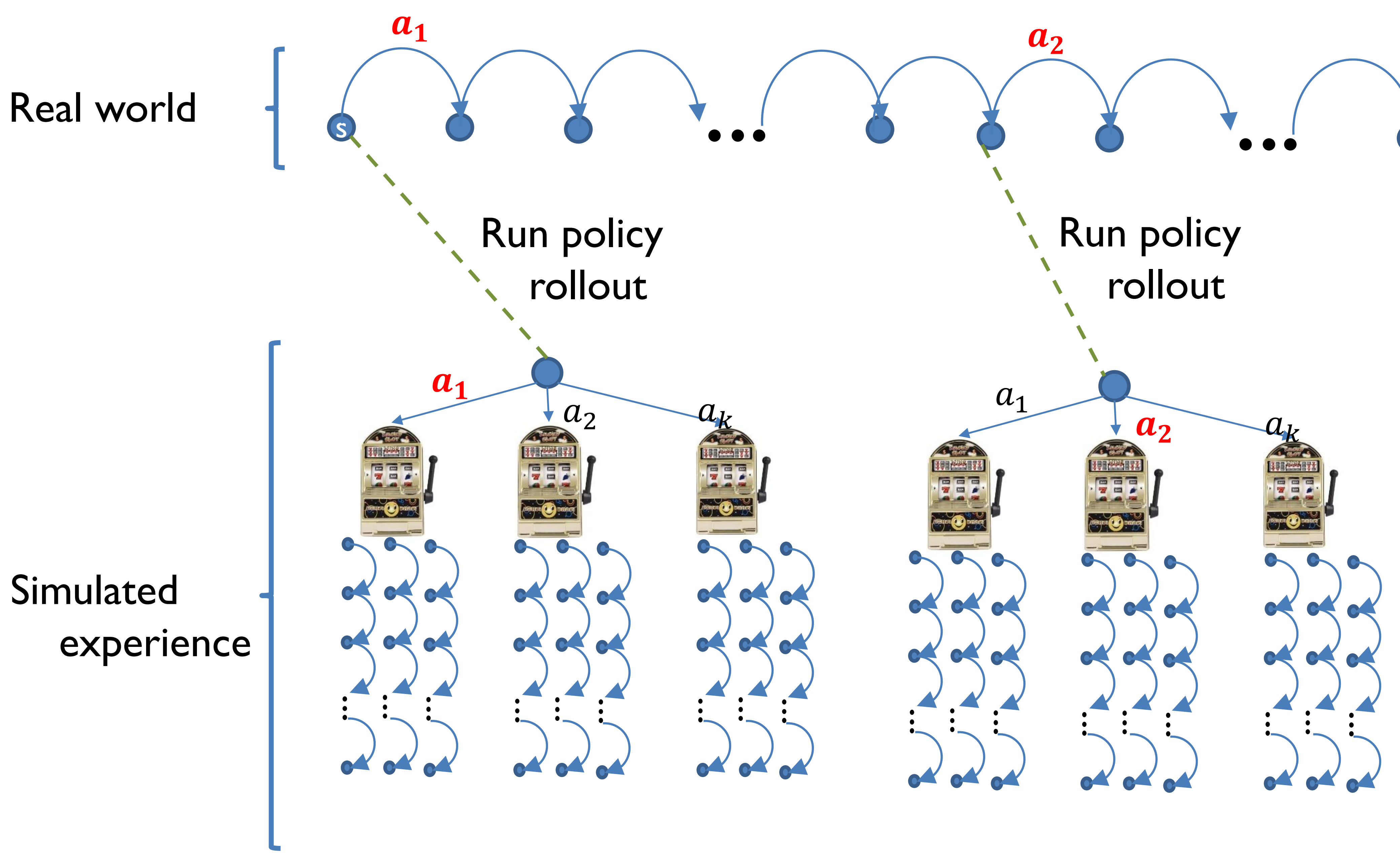
Policy Rollout algorithm

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

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



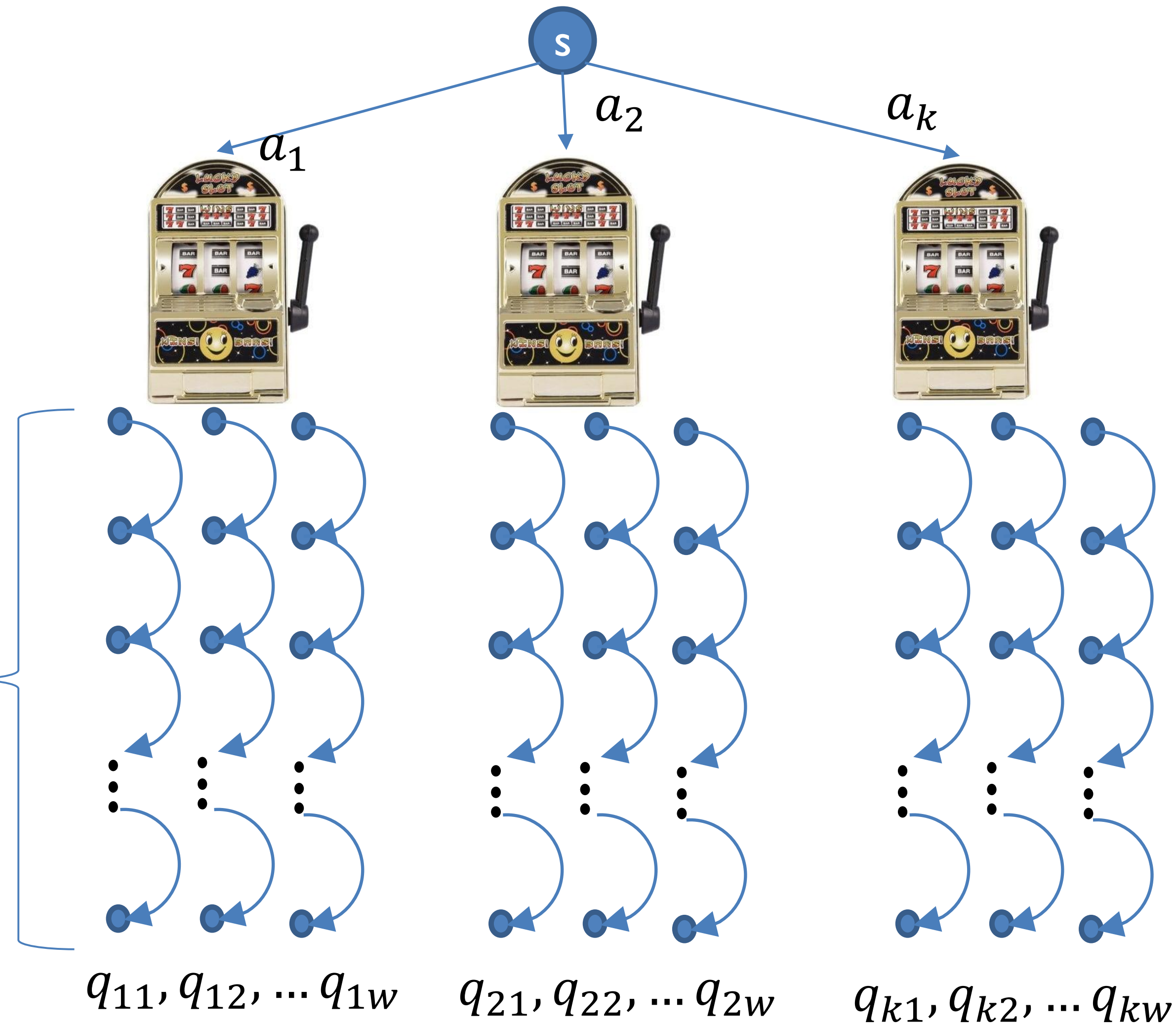
Executing Policy Rollout in the world



Policy Rollout: # of simulator calls

- For each action there is **w** calls to SimQ, each using **h** calls
- In total, **khw** calls to the simulator

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



Policy Rollout: PAC Guarantee

- Let a^* be the action that maximizes the true Q-function $Q_\pi(s,a)$.
- Let a' be the action return by policy rollout.
- Using the PAC result for single state, we get following:

If $w \geq \left(\frac{R_{max}}{\epsilon}\right)^2 \ln\frac{k}{\delta}$ then for all arms with probability $1-\delta$

$$|Q_\pi(s,a^*) - Q_\pi(s,a')| \leq \epsilon + \gamma^h V_{max}$$

Does this mean that the policy generated by the rollout will be close to the π' (from the Howard theorem)?

Policy Rollout: Quality

- How good is policy rollout compared to π'?

\- for fixed h and w there exists MDP such that rollout policy is arbitrarily worse than π'

- The MDP example is constructed for given parameters and is quite artificial.

\+ adding assumptions to the MDP, h and w can be chosen so that rollout quality is close to π'.

- Complicated

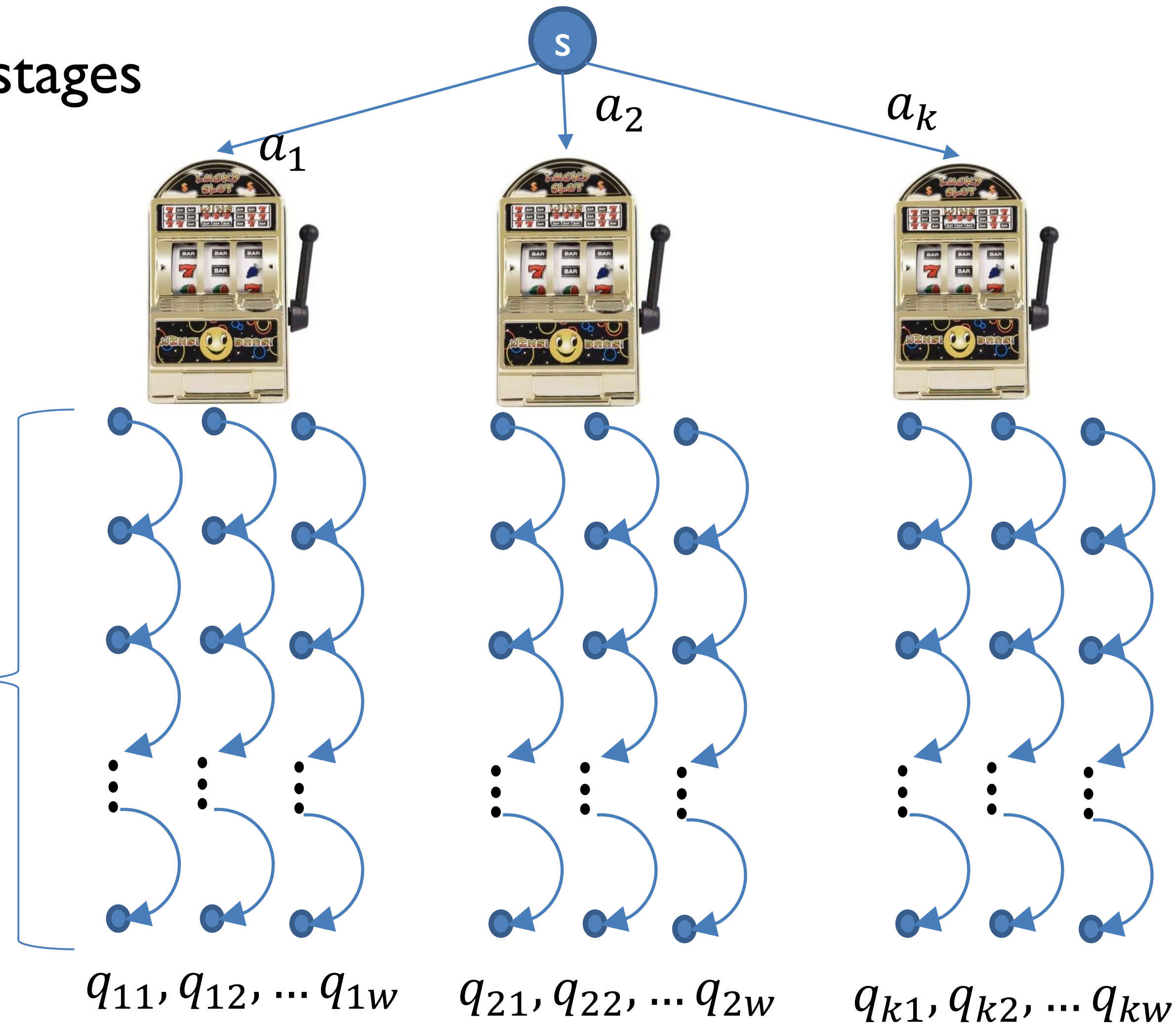
\+ h and w can be selected so that rollout is (approximately) no worse than π in any MDP

- So it will never hurt, only help

Multi-stage Rollout

- Two stage: compute rollout policy of "rollout of policy π"
- Requires $(\boldsymbol{khw})^2$ calls to the simulator
- Exponential in number of stages

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



Rollout summary

- Often, we can easily write simple policies
  - Dijkstra replan for robot Emil
  - Backgammon
  - Solitaire
  - Network routing policy
- **Policy rollout is general and easy way to improve such policies given simulator**
- This often provides substantial improvement:
  - Backghamon
  - Go
  - Solitaire
  - …

Rollout summary

- Policy Switching
  - Set of base policies, $\{\pi_1, \pi_2, \ldots \pi_M\}$
  - Instead of actions, try different policies in state S using Sim(s,π_i,h)
  - Works for any number of actions
- Single call to Rollout[π](s) approximates one iteration of policy iteration inialized at policy π

Example: Rollouts for Solitaire [Yan et al. Nips 2004]

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

Example: Rollouts for Solitaire [Yan et al. Nips 2004]

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: Rollouts for Solitaire [Yan et al. Nips 2004]

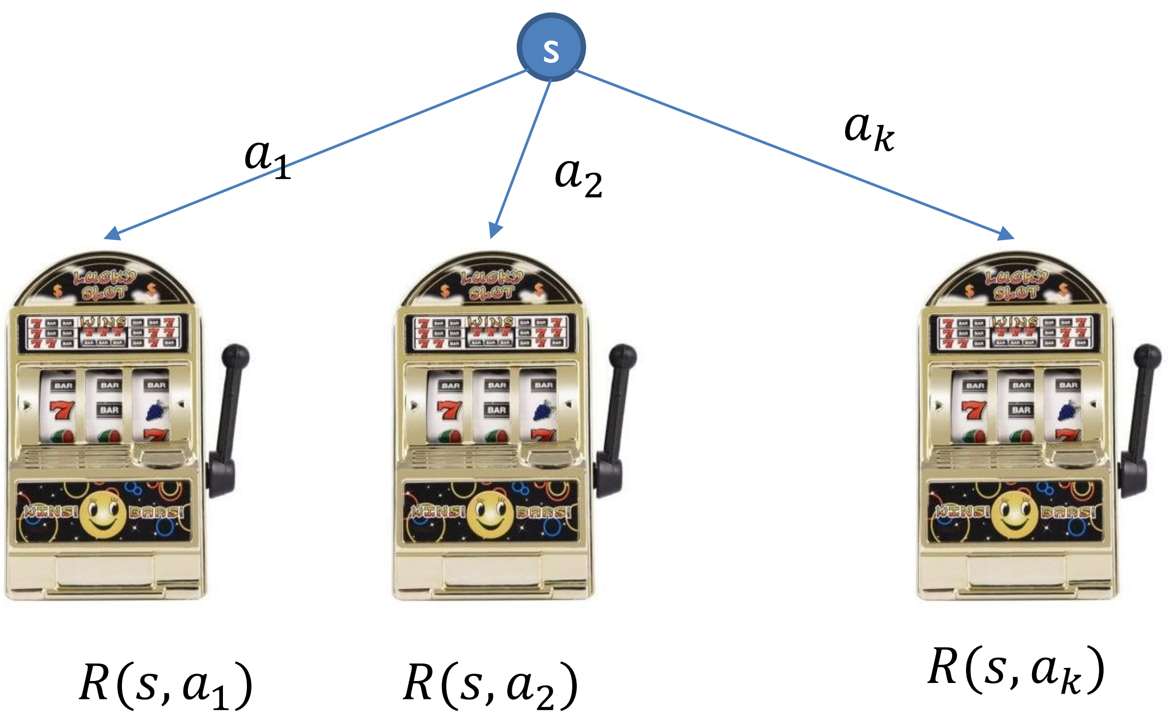
Player	Success Rate	Time/Game
Human Expert	36.6%	20 min
(naïve) Base Policy	13.05%	0.021 sec
1 rollout	31.20%	0.67 sec
2 rollout	47.6%	7.13 sec
3 rollout	56.83%	1.5 min
4 rollout	60.51%	18 min
5 rollout	70.20%	1 hour 45 min

Monte Carlo

- Non-Adaptive Monte-Carlo
  - Single state case (PAC Bandit)
  - Policy rollouts
- Adaptive Monte-Carlo
  - **Single state case (UCB Bandit)**
  - UCT MCTS

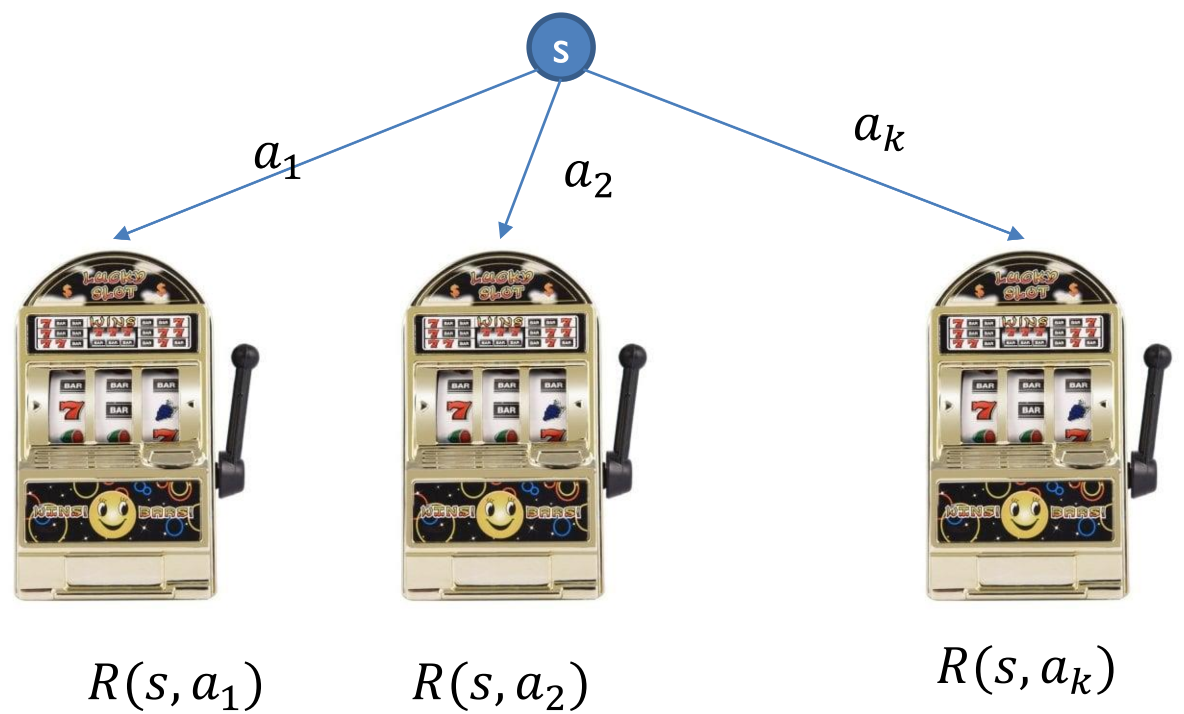
Multi-Armed bandit - PAC

- **Task: Select arm** that **probably** (prob. $1-\delta$) has **approximately** (within ϵ) the best expected reward.
- PAC = Probably Approximately Correct
- Use least possible number of runs.



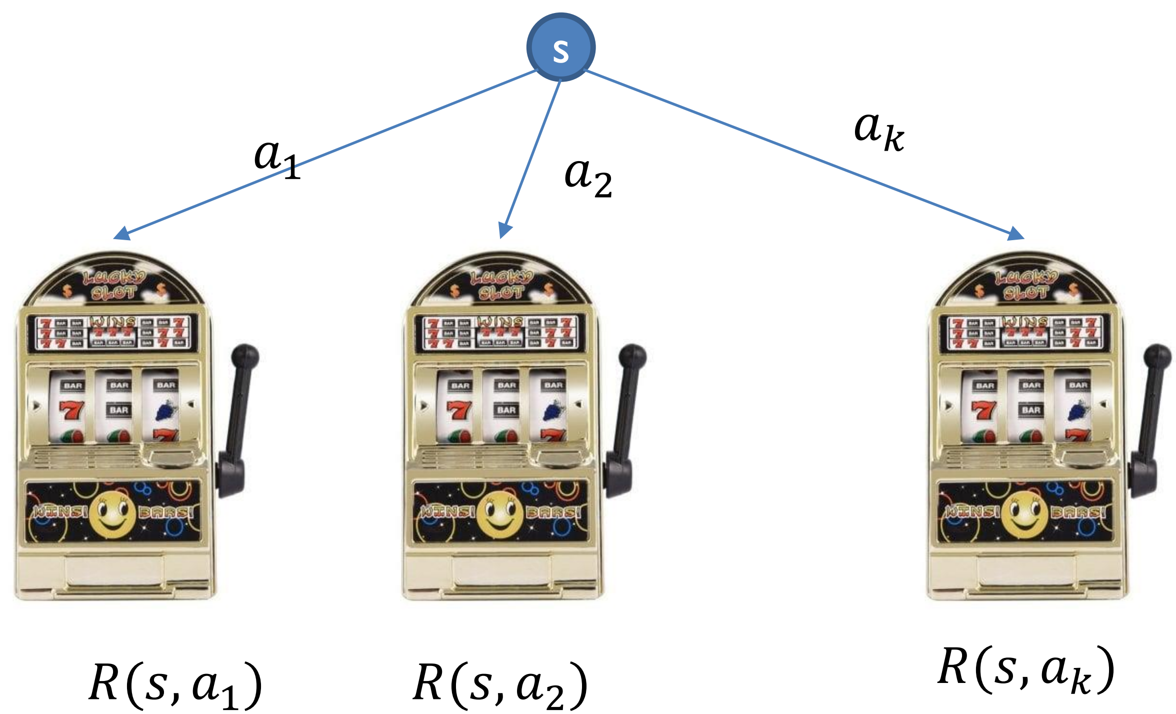
Multi-Armed bandit – Regret Minimization

- **Task:** find arm-pulling strategy such that the expected total reward at time n is close to the best possible.
  - Uniform Bandit – bad choice, wastes time with bad arms
  - Need to balance exploitation of good arms with exploration of poorly understood arms.



UCB Adaptive Bandit Algortihm

- **Task:** find arm-pulling strategy such that the expected total reward at time n is close to the best possible.
  - Uniform Bandit – bad choice, wastes time with bad arms
  - Need to balance exploitation of good arms with exploration of poorly understood arms.



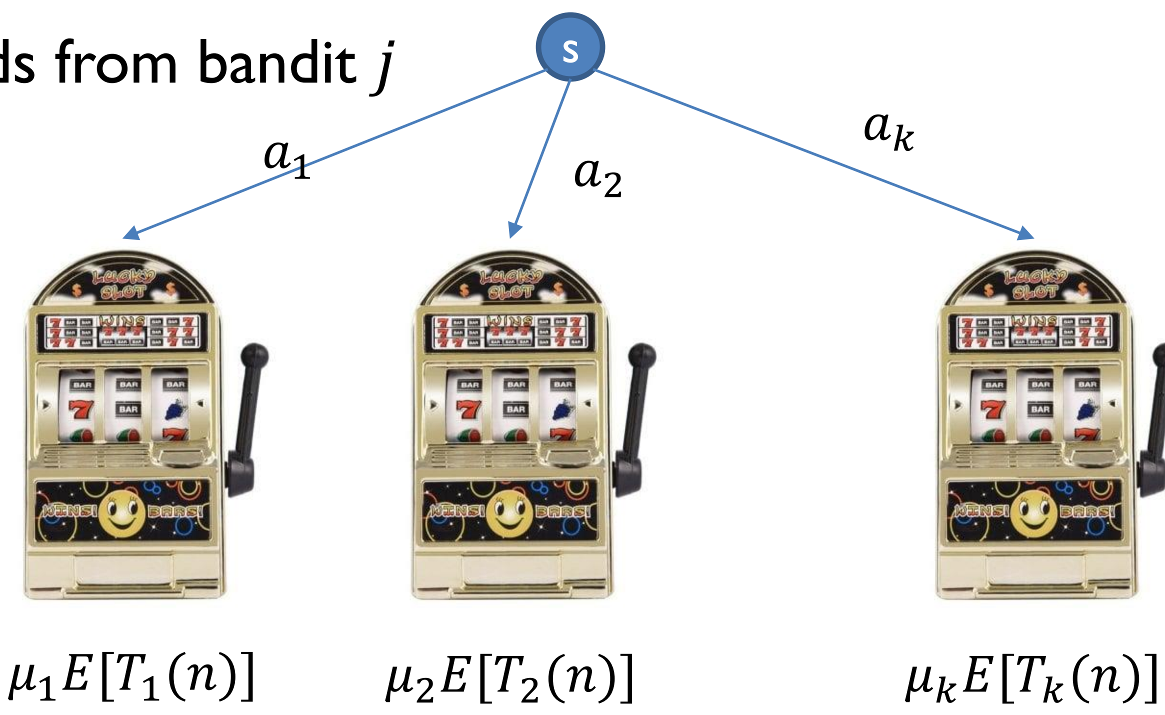
Regret

- Aiming at "reward as close as possible to the best reward" means we are minimizing **regret:**

$$R_n = \mu^* n - \sum_{j=1}^{k} \mu_j E[T_j(n)]$$

Where μ_j are the expected payoffs of arms, μ^* is the best expected payoff and $E[T_j(n)]$ is the expected number of pulls on arm j in total n pulls.

- $X_{j,1}, X_{j,2}$ ... = i.i.d r.v. of rewards from bandit j
- μ_j= expected value of X_j



Minimizing regret - UCB

- Upper Confidence Bounds [Auer er. al., 2002]:

$$UCB = \bar{X}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

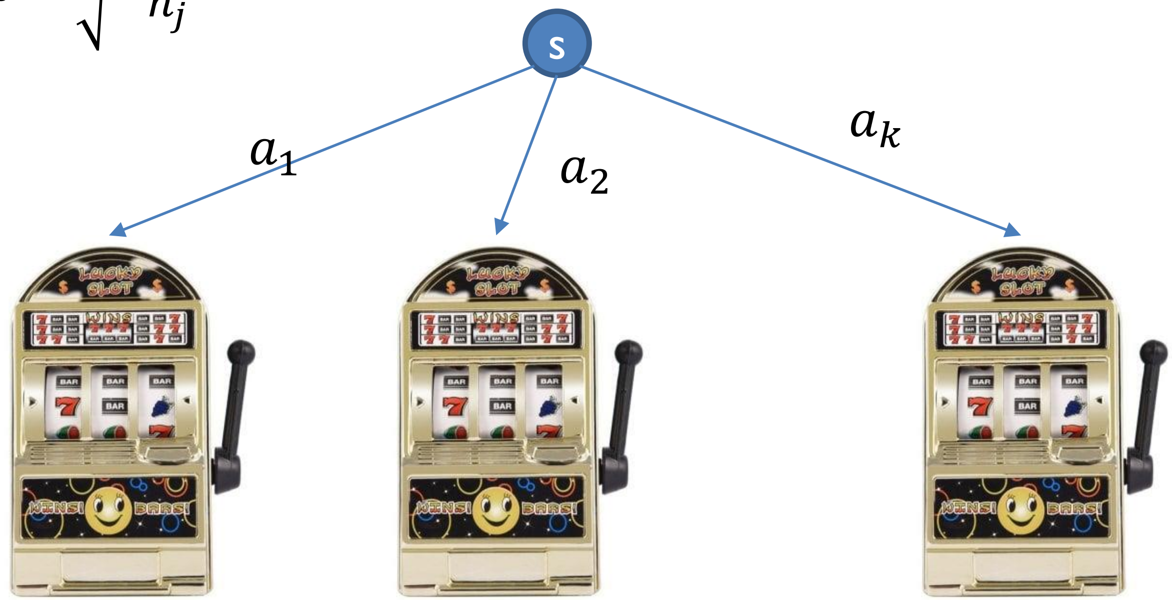
Exploitation

Exploration

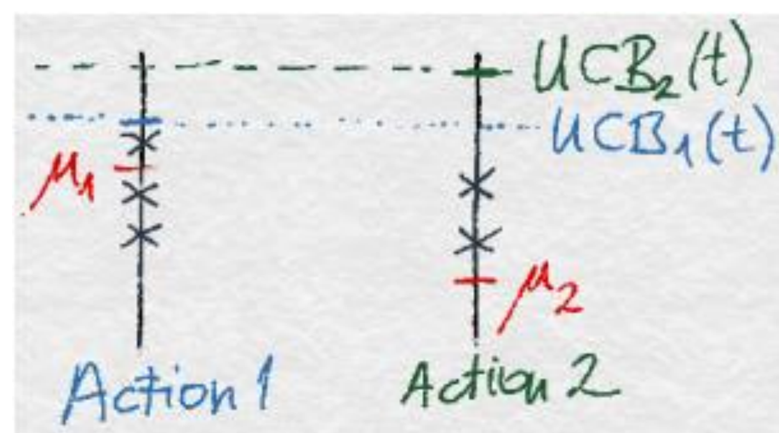
- When choosing arm, always select arm with highest UCB value
- $\bar{X}_j$ = mean of observed rewards, n = number of plays so far

UCB - Example

$$\bar{X}_j + \sqrt{\frac{2 \ln n}{n_j}}$$



- Play all arms once initially
- Then based on the formula



UCB - Example

$$\bar{X}_j + \sqrt{\frac{2\ln n}{n_j}}$$

- $\sqrt{\frac{2\ln n}{n_j}}$ is based of bound of the form $P\left(\bar{X}_j - E[X] \geq f(\sigma, n)\right) \leq \sigma$ (Remember PAC?)
- And σ is chosen to be time dependent (by n), goes to zero.

Excel example:

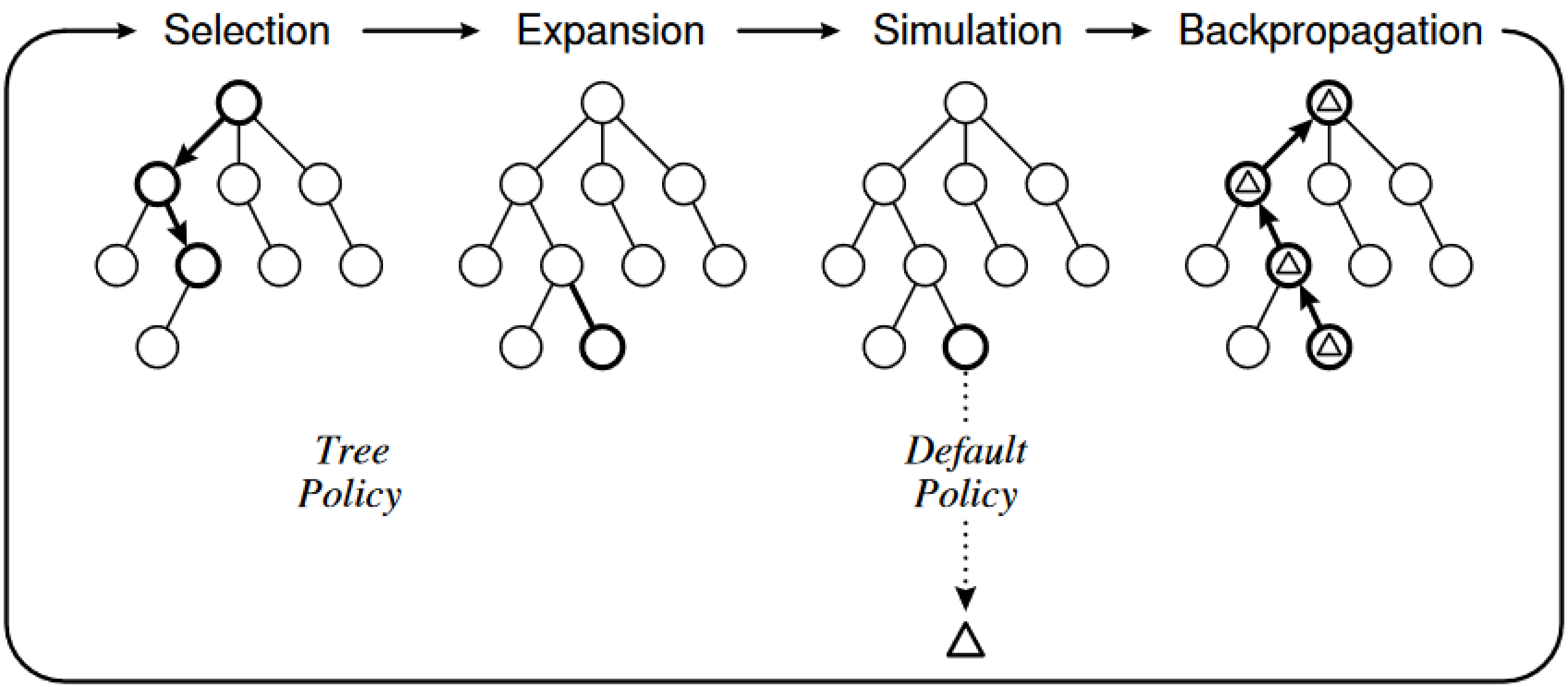
https://drive.google.com/open?id=1A9Kr-JDz_ZJlYOX3aFMrFaLUAPeAZV7Z

Monte Carlo

- Non-Adaptive Monte-Carlo
  - Single state case (PAC Bandit)
  - Policy rollouts
- Adaptive Monte-Carlo
  - Single state case (UCB Bandit)
  - **UCT MCTS**

Slides from Alan Fern, Dan Weld, Dan Klein, Luke Zettlmoyer

UCB for Trees = UCT



•Tree node:

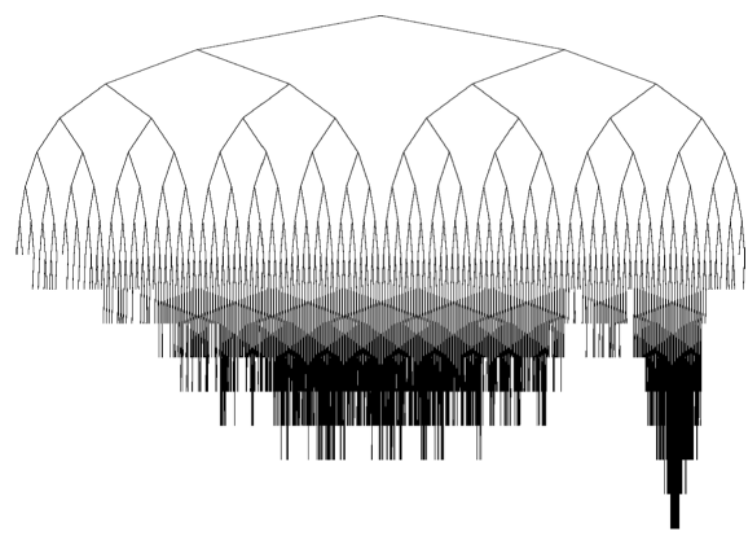
- Associated state,
- incoming action,
- number of visits,
- accumulated reward

•External slides by Michele Sebag:
https://drive.google.com/open?id=1ytp9l33_6WNe62qLAzV326iS4WmYeFpY

MCTS notes

- Aheuristic
  - Does not require any domain specific knowledge
  - Domain specific knowledge can provide significant speedups
- Anytime
  - Can return currently best action when stopped at any time
- Asymmetric
  - Tree is not explored fully
- MCTS = UCT? No consistency

  in the naming



[Arnaud et al., 2007]