Monte-Carlo Planning: Introduction and Bandit Basics

Alan Fern

Large Worlds

- We have considered basic model-based planning algorithms

- **Model-based planning**: assumes MDP model is available
  - Methods we learned so far are at least poly-time in the number of states and actions
  - Difficult to apply to large state and action spaces (though this is a rich research area)

- We will consider various methods for overcoming this issue

Approaches for Large Worlds

- <u>**Planning with compact MDP representations**</u>
  1. Define a language for compactly describing an MDP
     - MDP is exponentially larger than description
     - E.g. via Dynamic Bayesian Networks
  2. Design a planning algorithm that directly works with that language
- Scalability is still an issue
- Can be difficult to encode the problem you care about in a given language
- Study in last part of course

Approaches for Large Worlds

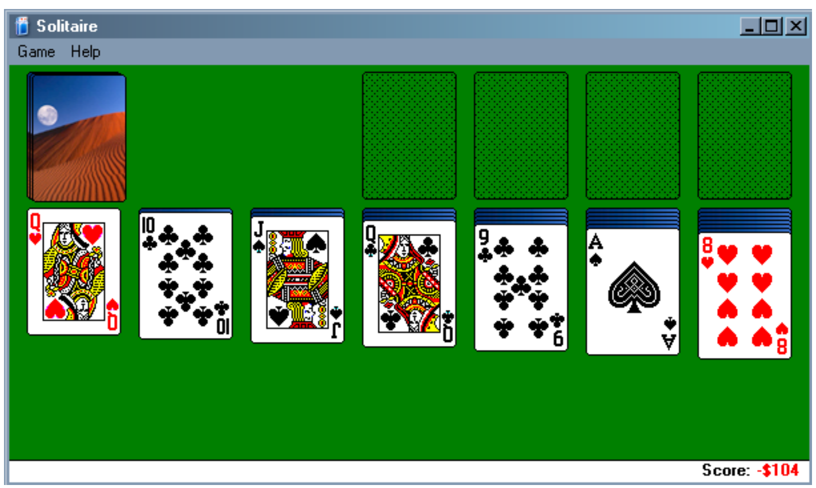
- **Reinforcement learning w/ function approx.**
  1. Have a learning agent directly interact with environment
  2. Learn a compact description of policy or value function

- Often works quite well for large problems
- Doesn’t fully exploit a simulator of the environment when available
- We will study reinforcement learning later in the course

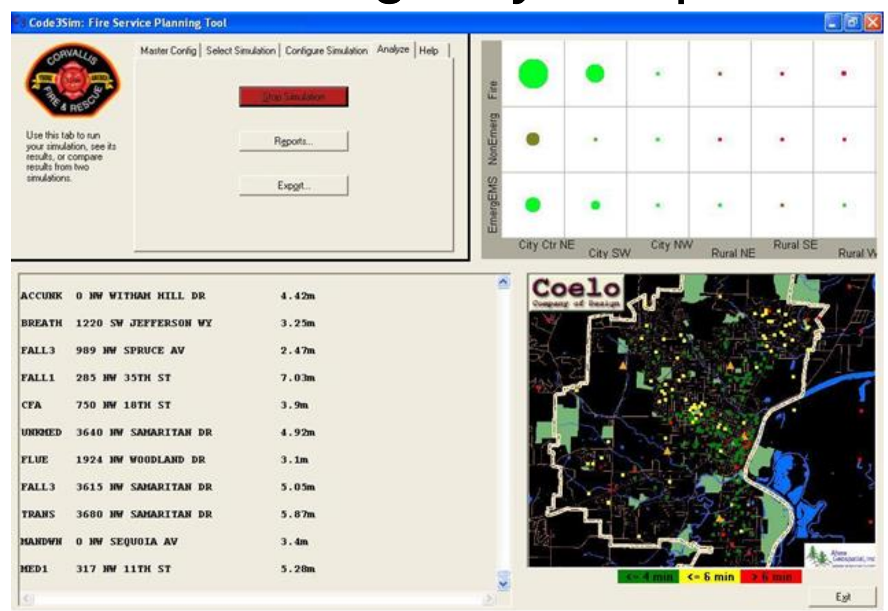
Approaches for Large Worlds: Monte-Carlo Planning

- Often a simulator of a planning domain is available or can be learned/estimated from data

Klondike Solitaire



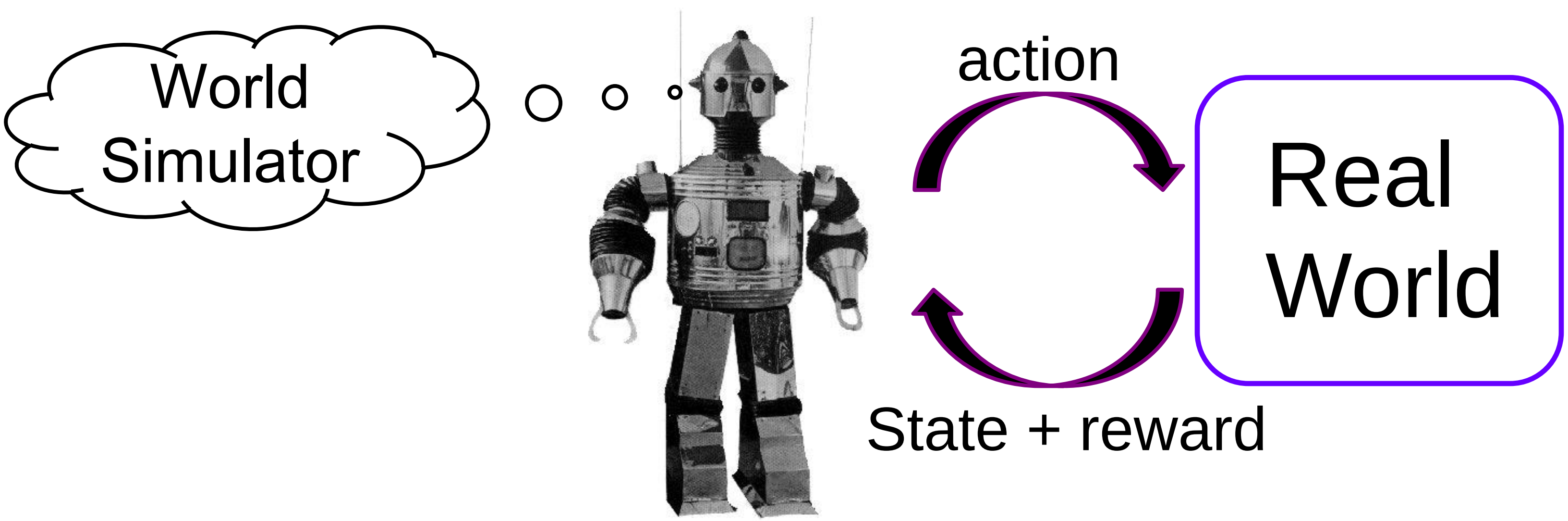
Fire & Emergency Response



Large Worlds: Monte-Carlo Approach

- Often a simulator of a planning domain is available or can be learned from data

- **Monte-Carlo Planning:** compute a good policy for an MDP by interacting with an MDP simulator



Example Domains with Simulators

- Traffic simulators
- Robotics simulators
- Military campaign simulators
- Computer network simulators
- Emergency planning simulators
  - large-scale disaster and municipal
- Forest Fire Simulator
- Board games / Video games
  - Go / RTS

In many cases Monte-Carlo techniques yield state-of-the-art performance. Even in domains where exact MDP models are available.

MDP: Simulation-Based Representation

- A simulation-based representation gives: S, A, R, T, I:
  - finite state set S   (|S|=n and is generally very large)
  - finite action set A (|A|=m and will assume is of reasonable size)
- |S| is too large to provide a matrix representation of R, T, and I (see next slide for I)
- A simulation based representation provides us with callable functions for R, T, and I.
  - Think of these as any other library function that you might call
- Our planning algorithms will operate by repeatedly calling those functions in an intelligent way

MDP: Simulation-Based Representation

- A simulation-based representation gives: S, A, R, T, I:
  - finite state set S (|S|=n and is generally very large)
  - finite action set A (|A|=m and will assume is of reasonable size)
  - Stochastic, real-valued, bounded reward function R(s,a) = r
    - Stochastically returns a reward **r** given input s and a
      (note: here rewards can depend on actions and can be stochastic)
  - Stochastic transition function T(s,a) = s' (i.e. a simulator)
    - Stochastically returns a state **s'** given input s and a
    - Probability of returning **s'** is dictated by Pr(s' | s,a) of MDP
  - Stochastic initial state function I.
    - Stochastically returns a state according to an initial state distribution

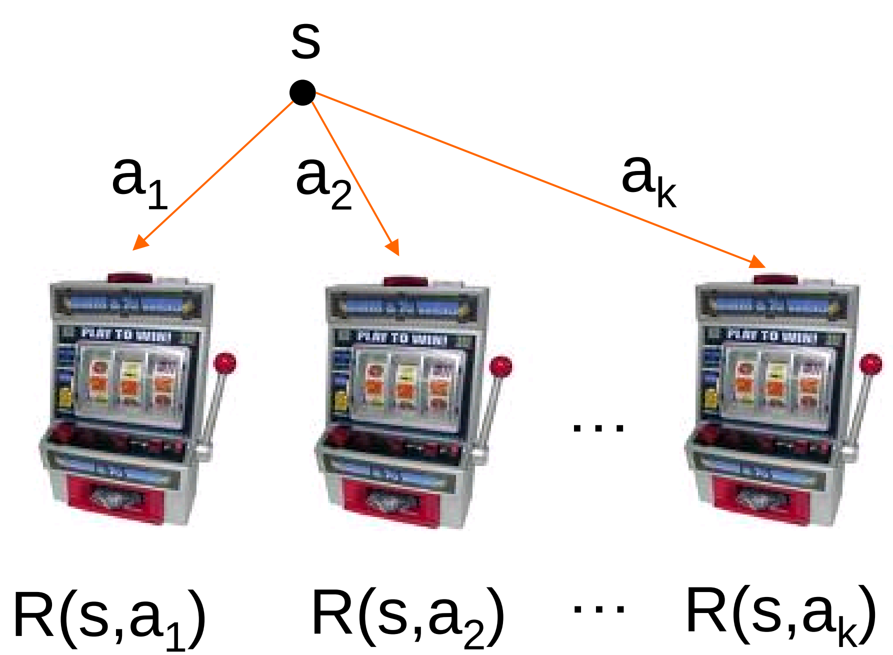
These stochastic functions can be implemented in any language!

Monte-Carlo Planning Outline

- Single State Case (multi-armed bandits)
  - A basic tool for other algorithms
- Monte-Carlo Policy Improvement
  - Policy rollout
  - Policy Switching
  - Approximate Policy Iteration
- Monte-Carlo Tree Search
  - Sparse Sampling
  - UCT and variants

Single State Monte-Carlo Planning

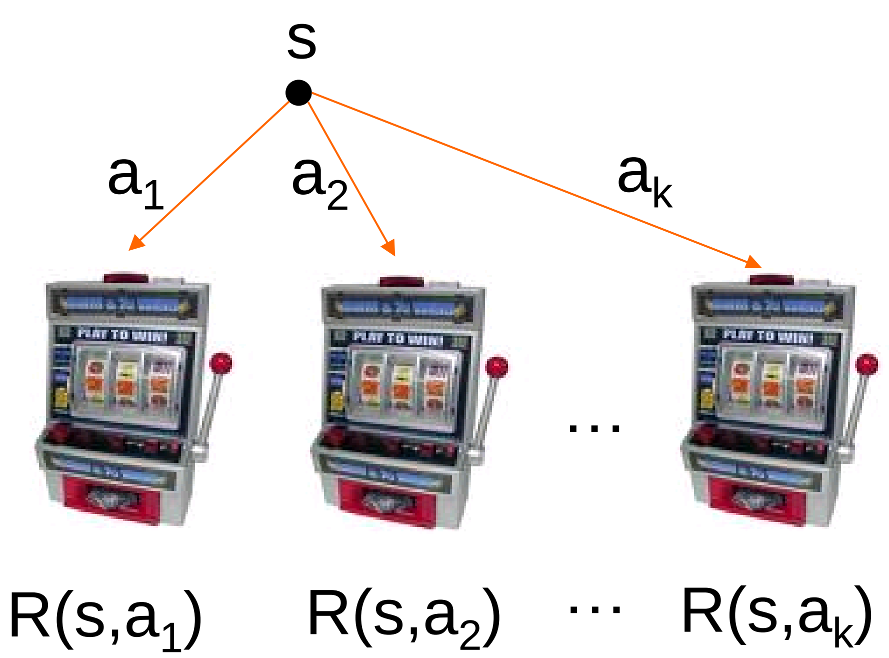
- Suppose MDP has a single state and k actions
  - Can sample rewards of actions using calls to simulator
  - Sampling action *a* is like pulling slot machine arm with random payoff function *R*(*s*,*a*)



Multi-Armed Bandit Problem

Single State Monte-Carlo Planning

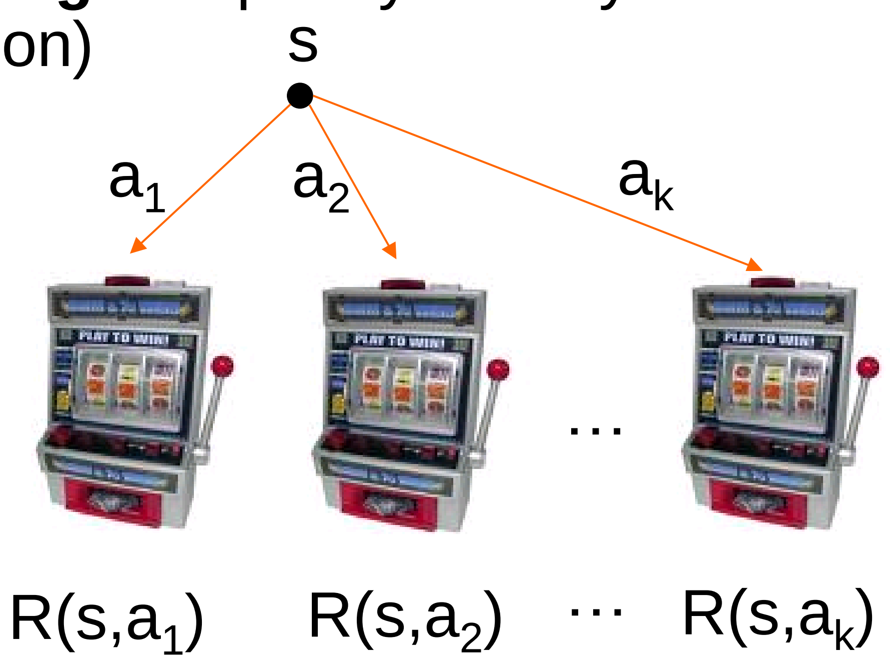
- Bandit problems arise in many situations
  - Clinical trials (arms correspond to treatments)
  - Ad placement (arms correspond to ad selections)



Multi-Armed Bandit Problem

Single State Monte-Carlo Planning

- We will consider three possible bandit objectives
  - **PAC Objective:** find a near optimal arm w/ high probability
  - **Cumulative Regret:** achieve near optimal cumulative reward over lifetime of pulling (in expectation)
  - **Simple Regret:** quickly identify arm with high reward (in expectation)



Multi-Armed Bandit Problem

Multi-Armed Bandits

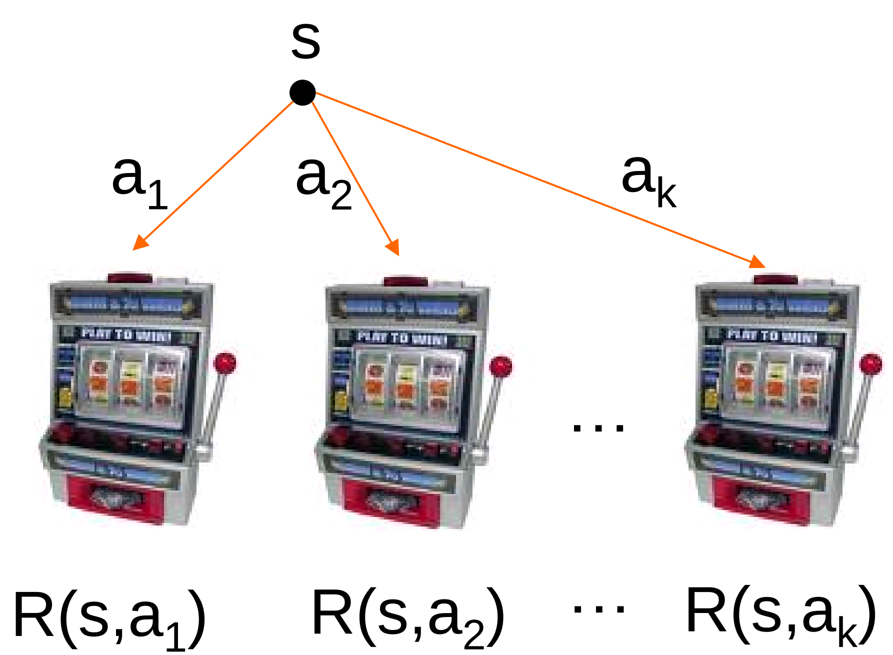
- Bandit algorithms are not just useful as components for multi-state Monte-Carlo planning
- Pure bandit problems arise in many applications
- Applicable whenever:
  - We have a set of independent options with unknown utilities
  - There is a cost for sampling options or a limit on total samples
  - Want to find the best option or maximize utility of our samples

Multi-Armed Bandits: Examples

- **Clinical Trials**
  - Arms = possible treatments
  - Arm Pulls = application of treatment to inidividual
  - Rewards = outcome of treatment
  - Objective = maximize cumulative reward = maximize benefit to trial population (or find best treatment quickly)
- **Online Advertising**
  - Arms = different ads/ad-types for a web page
  - Arm Pulls = displaying an ad upon a page access
  - Rewards = click through
  - Objective = maximize cumulative reward = maximum clicks (or find best add quickly)

PAC Bandit Objective: Informal

- **Probably Approximately Correct (PAC)**
  - Select an arm that probably (w/ high probability) has approximately the best expected reward
  - Design an algorithm that uses as few simulator calls (or pulls) as possible to guarantee this



Multi-Armed Bandit Problem

PAC Bandit Algorithms

- Let k be the number of arms, R_{max} be an upper bound on reward, and $R^* = \max_i E[R(s,a_i)]$ (i.e. R^* is the best arm reward in expectation)

Definition (Efficient PAC Bandit Algorithm): An algorithm ALG is an efficient PAC bandit algorithm iff for any multi-armed bandit problem, for any $0<\delta<1$ and any $0<\varepsilon<1$ (these are inputs to ALG), ALG pulls a number of arms that is **polynomial in $1/\varepsilon$, $1/\delta$, *k*, *and* R_{max}** and returns an arm index j such that with probability at least $1-\delta$

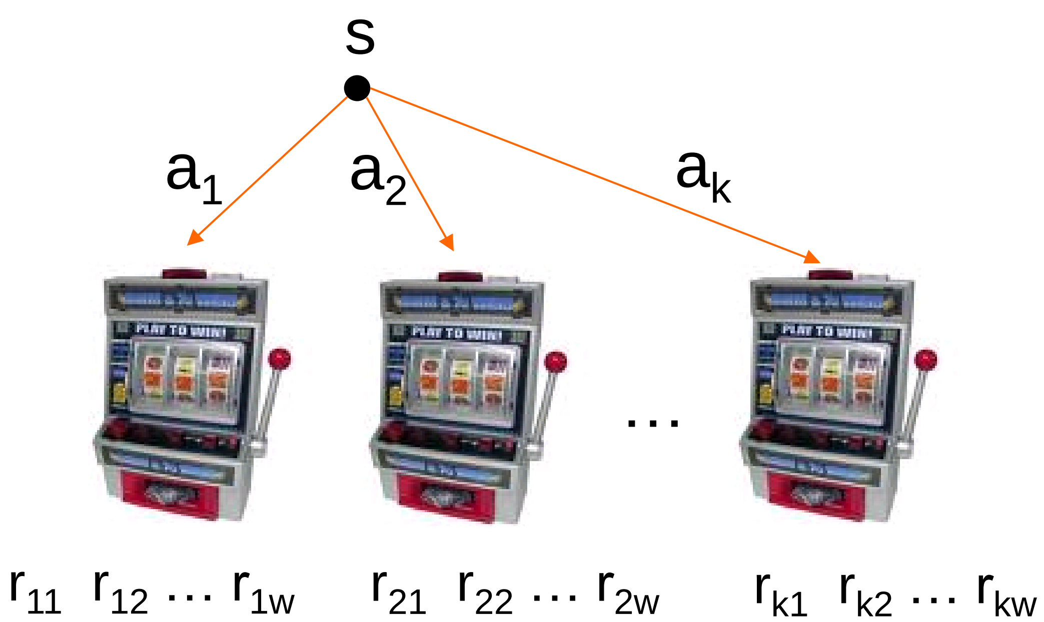
$$R^* - E[R(s,a_j)] \le \varepsilon$$

- Such an algorithm is efficient in terms of # of arm pulls, and is probably (with probability $1-\delta$) approximately correct (picks an arm with expected reward within ε of optimal).

UniformBandit Algorithm

Even-Dar, E., Mannor, S., & Mansour, Y. (2002). PAC bounds for multi-armed bandit and Markov decision processes. In *Computational Learning Theory*

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



Can we make this an efficient PAC bandit algorithm?

Aside: Additive Chernoff Bound

- Let R be a random variable with maximum absolute value Z. An let r_i 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

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

Equivalent Statement:

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

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

Aside: Coin Flip Example

- Suppose we have a coin with probability of heads equal to p.
- Let X be a random variable where X=1 if the coin flip gives heads and zero otherwise. (so Z from bound is 1)

$$E[X] = 1*p + 0*(1-p) = p$$

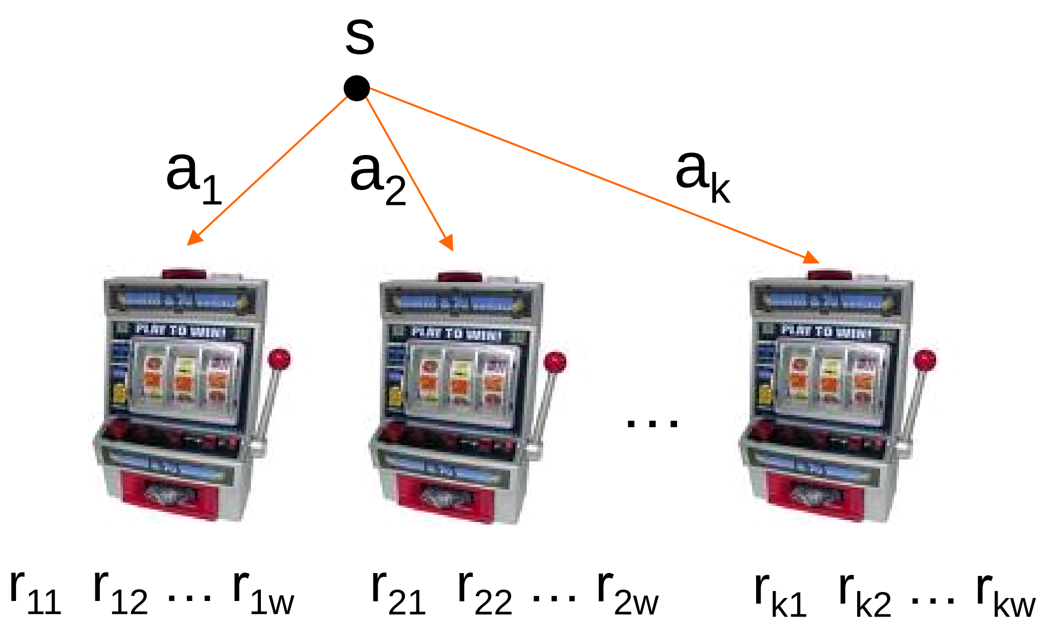
- After flipping a coin w times we can estimate the heads prob. by average of x_i.
- The Chernoff bound tells us that this estimate converges exponentially fast to the true mean (coin bias) p.

$$\Pr\left(\left|p - \tfrac{1}{w}\sum_{i=1}^{w} x_i\right| \geq \varepsilon\right) \leq \exp\left(-\varepsilon^2 w\right)$$

UniformBandit Algorithm

Even-Dar, E., Mannor, S., & Mansour, Y. (2002). PAC bounds for multi-armed bandit and Markov decision processes. In *Computational Learning Theory*

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



Can we make this an efficient PAC bandit algorithm?

UniformBandit PAC Bound

- For a single bandit arm the Chernoff bound says:

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

$$\left| E[R(s,a_i)] - \tfrac{1}{w}\sum_{j=1}^{w} r_{ij} \right| \le R_{\max}\sqrt{\tfrac{1}{w}\ln\tfrac{1}{\delta'}}$$

- Bounding the error by ε gives:

$$R_{\max}\sqrt{\tfrac{1}{w}\ln\tfrac{1}{\delta'}} \le \varepsilon \quad \text{or equivalently} \quad w \ge \left(\frac{R_{\max}}{\varepsilon}\right)^2 \ln\tfrac{1}{\delta'}$$

- Thus, using this many samples for a single arm will guarantee an ε-accurate estimate with probability at least $1-\delta'$

UniformBandit PAC Bound

- So we see that with $w \geq \left(\frac{R_{\max}}{\varepsilon}\right)^2 \ln \frac{1}{\delta'}$ samples per arm, there is no more than a δ' probability that an individual arm's estimate will **not** be ε-accurate
  - But we want to bound the probability of any arm being inaccurate

The **union bound** says that for *k* events, the probability that at least one event occurs is bounded by the sum of individual probabilities

$$\Pr(A_1 \; or \; A_2 \; or \cdots or \; A_k) \leq \sum_{i=1}^{k} \Pr(A_k)$$

- Using the above # samples per arm and the union bound (with events being "arm *i* is not ε-accurate") there is no more than $k\delta'$ probability of any arm not being ε-accurate

- Setting $\delta' = \frac{\delta}{k}$ all arms are ε-accurate with prob. at least $1 - \delta$

UniformBandit PAC Bound

Putting everything together we get:

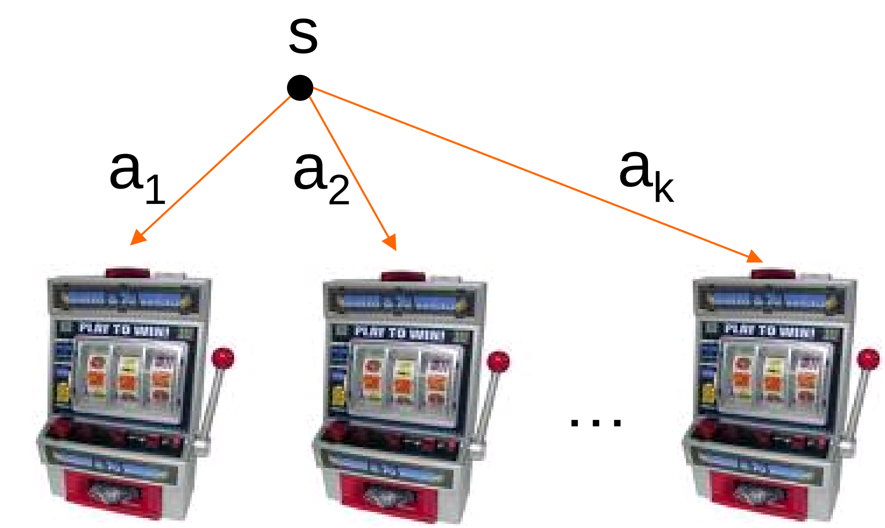
If $w \geq \left(\frac{R_{\max}}{\varepsilon}\right)^2 \ln\frac{k}{\delta}$ then for all arms simultaneously

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

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

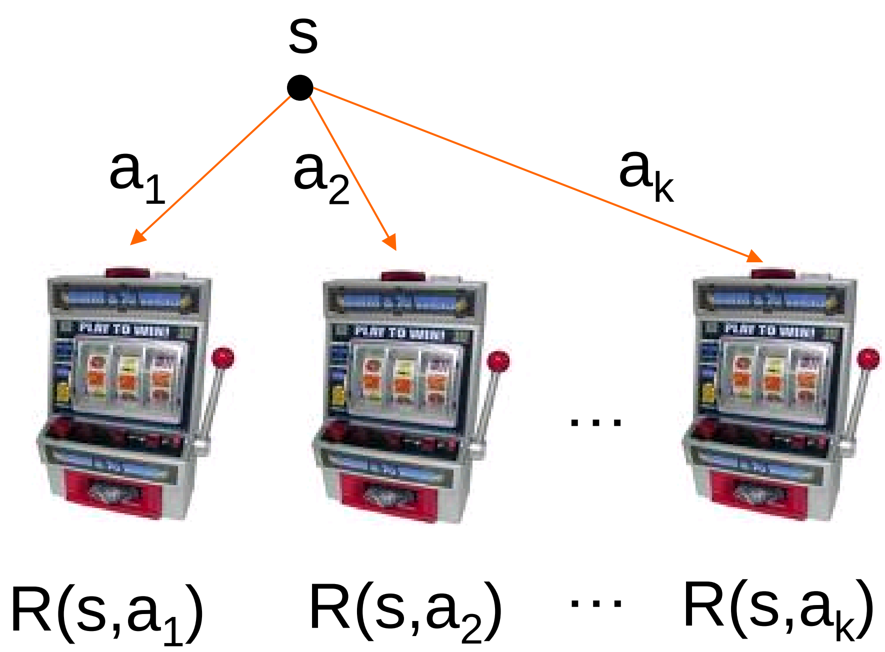
Simulator Calls for UniformBandit



$R(s,a_1)$ $R(s,a_2)$ … $R(s,a_k)$

- Total simulator calls for PAC: $k \cdot w = \left(\frac{R_{max}}{\varepsilon}\right)^2 k \ln \frac{k}{\delta}$
  - ➢ So we have an **efficient** PAC algorithm
  - ➢ Can we do better than this?

Non-Uniform Sampling



- If an arm is really bad, we should be able to eliminate it from consideration early on
- **Idea:** try to allocate more pulls to arms that appear more promising

Median Elimination Algorithm

Even-Dar, E., Mannor, S., & Mansour, Y. (2002). PAC bounds for multi-armed bandit and Markov decision processes. In *Computational Learning Theory*

Median Elimination

A = set of all arms

For i = 1 to …..

    Pull each arm in A $\mathbf{w_i}$ times

    m = median of the average rewards of the arms in A

    A = A – {arms with average reward less than m}

    If |A| = 1 then return the arm in A

Eliminates half of the arms each round.

How to set the $\mathbf{w_i}$ to get PAC guarantee?

Median Elimination (proof not covered)

- Theoretical values used by Median Elimination:

$$w_i = \frac{4}{\epsilon_i^2} \ln \frac{3}{\delta_i} \qquad \epsilon_i = \left(\frac{3}{4}\right)^{i-1} \cdot \frac{\epsilon}{4} \qquad \delta_i = \frac{\delta}{2^i}$$

Theorem: Median Elimination is a PAC algorithm and uses a number of pulls that is at most $O\left(\frac{k}{\varepsilon^2} \ln \frac{1}{\delta}\right)$

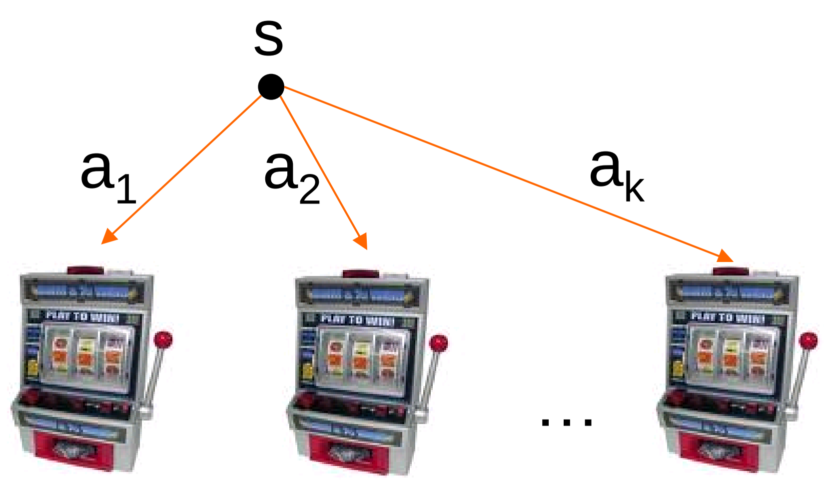
Compare to $O\left(\frac{k}{\varepsilon^2} \ln \frac{k}{\delta}\right)$ for UniformBandit

PAC Summary

- Median Elimination uses O(log(k)) fewer pulls than Uniform
  - Known to be asymptotically optimal (no PAC algorithm can use fewer pulls in worst case)
- PAC objective is sometimes awkward in practice
  - Sometimes we don't know how many pulls we will have
  - Sometimes we can't control how many pulls we get
  - Selecting ϵ and δ can be quite arbitrary
- Cumulative & simple regret partly address this

Cumulative Regret Objective

- **Problem:** find arm-pulling strategy such that the expected total reward at time *n* is close to the best possible (one pull per time step)
  - Optimal (in expectation) is to pull optimal arm *n* times
  - UniformBandit is poor choice --- waste time on bad arms
  - Must balance **exploring** machines to find good payoffs and **exploiting** current knowledge



Cumulative Regret Objective

- Theoretical results are often about "expected cumulative regret" of an arm pulling strategy.

- **Protocol:** At time step n the algorithm picks an arm a_n based on what it has seen so far and receives reward r_n *(a_n and r_n are random variables)*.

- **Expected Cumulative Regret ($\boldsymbol{E[Reg_n]}$):** difference between optimal expected cummulative reward and expected cumulative reward of our strategy at time n

$$E[Reg_n] = n \cdot R^* - \sum_{i=1}^{n} E[r_n]$$

UCB Algorithm for Minimizing Cumulative Regret

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, *47*(2), 235-256.

- Q(a) : average reward for trying action a (in our single state *s*) so far
- n(a) : number of pulls of arm a so far
- Action choice by UCB after n pulls:

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

- Assumes rewards in [0,1]. We can always normalize if we know max value.

UCB: Bounded Sub-Optimality

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

Value Term:
favors actions that looked good historically

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

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

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

where Δ_a is the sub-optimality of arm **a**

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

UCB Performance Guarantee
[Auer, Cesa-Bianchi, & Fischer, 2002]

Theorem: The expected cumulative regret of UCB $E[Reg_n]$ after *n* arm pulls is bounded by O(log *n*)

- Is this good?

Yes. The average per-step regret is $O\left(\frac{\log(n)}{n}\right)$

Theorem: No algorithm can achieve a better expected regret (up to constant factors)

What Else ….

- UCB is great when we care about cumulative regret
- But, sometimes all we care about is finding a good arm quickly
- This is similar to the PAC objective, but:
  - The PAC algorithms required precise knowledge of or control of # pulls
  - We would like to be able to stop at any time and get a good result with some guarantees on expected performance
- “Simple regret” is an appropriate objective in these cases

Simple Regret Objective

- **Protocol:** At time step n the algorithm picks an "exploration" arm a_n to pull and observes reward r_n and also picks an arm index it thinks is best j_n ($a_n,\ j_n\ and\ r_n$ *are random variables)*.
  - If interrupted at time n the algorithm returns j_n.

- **Expected Simple Regret ($\boldsymbol{E[SReg_n]}$):** difference between R^* and expected reward of arm j_n selected by our strategy at time n

$$E[SReg_n] = R^* - E[R(a_{j_n})]$$

Simple Regret Objective

- What about UCB for simple regret?
  - Intuitively we might think UCB puts too much emphasis on pulling the best arm
  - After an arm starts looking good, we might be better off trying figure out if there is indeed a better arm

Theorem: The expected simple regret of UCB after *n* arm pulls is upper bounded by O(n^{-c}) for a constant c.

Seems good, but we can do much better in theory.

Incremental Uniform (or Round Robin)

Bubeck, S., Munos, R., & Stoltz, G. (2011). Pure exploration in finitely-armed and continuous-armed bandits. Theoretical Computer Science, 412(19), 1832-1852

Algorithm:

- At round n pull arm with index (k mod n) + 1
- At round n return arm (if asked) with largest average reward

Theorem: The expected simple regret of Uniform after *n* arm pulls is upper bounded by O(e^{-cn}) for a constant c.

- This bound is exponentially decreasing in n!

  Compared to polynomially for UCB O(n^{-c}).

Can we do better?

Tolpin, D. & Shimony, S, E. (2012). MCTS Based on Simple Regret. *AAAI Conference on Artificial Intelligence.*

Algorithm ϵ-Greedy **:** (parameter $0 < \epsilon < 1$)

- At round n, with probability ϵ pull arm with best average reward so far, otherwise pull one of the other arms at random.
- At round n return arm (if asked) with largest average reward

Theorem: The expected simple regret of ϵ-Greedy for $\epsilon = 0.5$ after *n* arm pulls is upper bounded by O(e^{-cn}) for a constant c that is larger than the constant for Uniform
(this holds for "large enough" n).

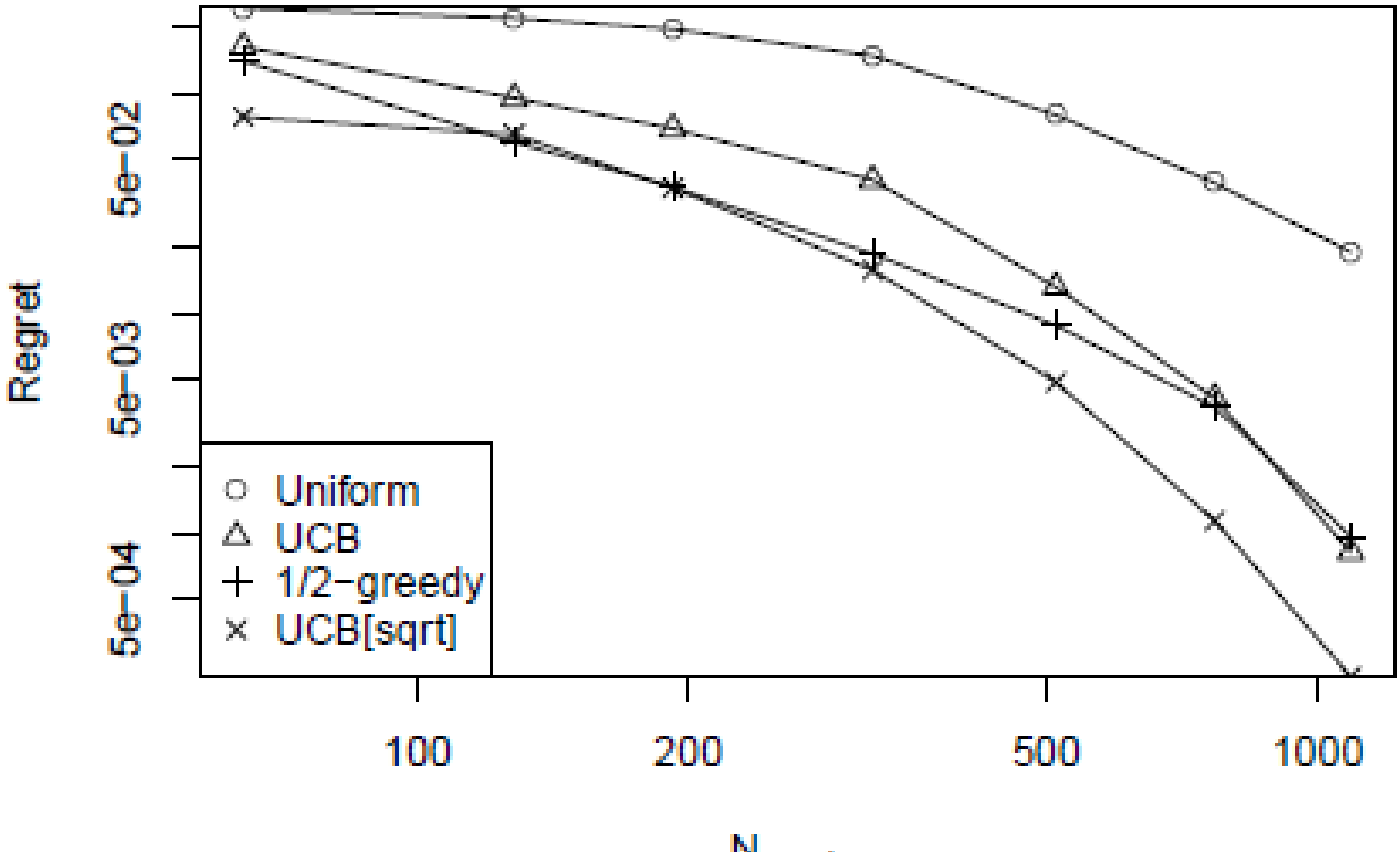
Summary of Bandits in Theory

- PAC Objective:
  - **UniformBandit** is a simple PAC algorithm
  - **MedianElimination** improves by a factor of log(k) and is optimal up to constant factors
- Cumulative Regret:
  - **Uniform** is very bad!
  - **UCB** is optimal (up to constant factors)
- Simple Regret:
  - **UCB** shown to reduce regret at polynomial rate
  - **Uniform** reduces at an exponential rate
  - **0.5-Greedy** may have even better exponential rate

Theory vs. Practice

- The established theoretical relationships among bandit algorithms have often been useful in predicting empirical relationships.
- But not always ….

Theory vs. Practice



b. regret vs. number of samples