# STA 4273: Minimizing Expectations

## Lecture 10 - Search and policy optimization

Chris J. Maddison

University of Toronto

# Announcements

- Do you want your slides and colab shared?
- Extension on the final project report. Now due April 14.

# Today

- Planning in MDPs.
  - Given access to the transition distribution of the MDP (i.e., assume the ability to sample as many transitions as you want starting in any state, action pair) , can we compute an optimal action?
- Monte Carlo Tree Search.
- But first, we will review a classic problem: multi-armed bandits.

# Multi-armed bandit problem

Imagine a row of slot machines with different expected payouts.

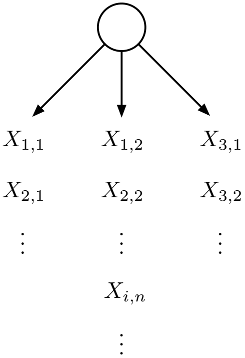
You have limited money, how do you pick which ones to play?

# Multi-armed bandit problem

- Classic exploration vs. exploitation tradeoff.
  - You need to explore to get an estimate of the expected payoffs, but this costs you money.
  - When should you switch to exploiting your knowledge, i.e., playing just what you think is the best machine?
- Classical problem, but insights from this model are core to efficient algorithms for planning in MDPs.

# Multi-armed bandit problem

- Multiple rounds.
- Each round $n$, your algorithm picks 1 of $K$ arms.
- If you pick arm $i$ on round $n$, you receive a random reward $X_{i,n} \in [0, 1]$
  - For each $i$, $X_{i,n}$ are i.i.d. $X_{i,n}$ are mutually independent for all $i, n$.



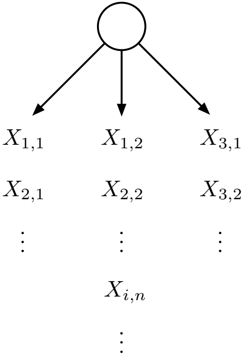
# Multi-armed bandit problem

- Let $\mu_i = \mathbb{E}[X_{i,n}]$. We want to achieve an expected reward of $\mu^* = \max_i \mu_i$.
- To quantify the optimal algorithm for $n$ rounds we want to minimize its regret

$$\mathbb{E}[R(n)] = n\mu^* - \sum_{i=1}^{K} \mathbb{E}[N_i(n)]\mu_i$$

where

$$N_i(n) = \sum_{t=1}^{n} \mathbb{I}[\,\text{pulled arm } i \text{ in round } t\,].$$



# *Non, je ne regrette rien*

- Lai and Robbins (1985) proved that any algorithm, which performs "consistently" across a family of bandit problems, must pull sub-optimal arms logarithmically many times, i.e., for $i \neq \arg\max_i \mu_i$,

$$\mathbb{E}[N_i(n)] = \Omega(\log n)$$

- They also gave an algorithm that achieved this and has an overall logarithmic regret bound (with instance dependent constants):

$$\mathbb{E}[R(n)] = \mathcal{O}(\log n)$$

# An upper confidence bound algorithm (UCB1)

UCB1 (Auer et al., 2002) is a modern simple version that achieves this logarithmic regret:

1. Initialize $\bar{X}_{i,0} = \infty$.
2. For each round $n$ pull the following arm:

$$\arg\max_i \left\{ \frac{\bar{X}_{i,n}}{N_i(n)} + \sqrt{\frac{2 \log n}{N_i(n)}} \right\}$$

3. Update $\bar{X}_{i,n} = \sum_{t=1}^{n} X_{i,t} \mathbb{I}[\,\text{pulled arm } i \text{ in round } t\,]$

# Multi-armed bandits and UCB1

$$\arg\max_i \left\{ \frac{\bar{X}_{i,n}}{N_i(n)} + \sqrt{\frac{2\log n}{N_i(n)}} \right\}$$

- UCB1 (Auer et al., 2002) has a very intuitive interpretation.
  - Exploration term is balanced by exploitation term.
- UCB1 keeps visiting all arms forever, but eventually pulls the optimal arm exponentially many times.
  - Must be true if suboptimal arms are pulled logarithmically many times.
  - Solves the exploration-exploitation tradeoff.

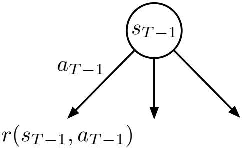
Can we use this to find the optimal policy of an MDP?

# Planning in an MDP

- You are an agent in an finite-horizon $T$, finite state space, finite action space MDP.
- Suppose you have your own model $p(s'|s, a)$ of the transition function that you can call as many times as you want.
  - Let's assume it's deterministic.
- Can you use this model to do lookahead planning to compute good moves? Let's consider some special cases.

# Planning in an MDP

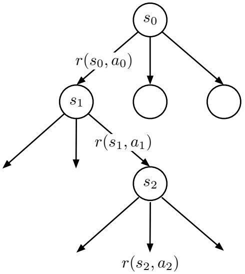
- Suppose you are in state $s$ in the final time step $T$.
- This is just a bandit problem! Taking an action is like pulling an arm.
- Idea: Run UCB1 for $n$ rounds, finally take the action that was most pulled.



# Planning in an MDP

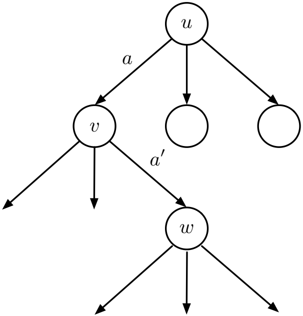
- Suppose you are in state $s$, but it is *not* the final time step.
- We can still think of this as a type of bandit problem.
- Taking an action is like pulling an arm that returns an immediate reward $r(s, a)$ and *a new bandit problem* over the future return.

$$r(s_0, a_0) + \gamma \sum_{t=1}^{T-1} \gamma^{t-1} r(s_t, a_t)$$



# Upper confidence trees (UCT)

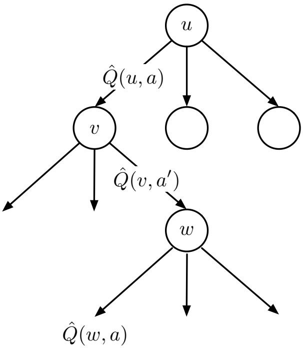
- The upper confidence trees (UCT, Kocsis & Szepesvári, 2006) algorithm takes this recursive perspective.
- Organize sequences of actions $(a_0, \ldots, a_{T-1})$ into a tree.
- Nodes $v$ represent action sequences with a common prefix and have a state $s(v)$ associated with them.



# Upper confidence trees (UCT)

For each explored node $v$, UCT maintains

- Estimate $\hat{Q}(v, a)$ of the optimal state-action value function $Q^*(s(v), a)$ at each node
- The count $N(v, a)$ of times action $a$ was chosen in node $v$.
  - Let $N(v) = \sum_a N(v, a)$.



# Upper confidence trees (UCT)

```
function MonteCarloPlanning(state)
repeat
    search(state, 0)
until Timeout
return bestAction(state,0)

function search(state, depth)
if Terminal(state) then return 0
if Leaf(state, d) then return Evaluate(state)
action := selectAction(state, depth)
(nextstate, reward) := simulateAction(state, action)
q := reward + γ search(nextstate, depth + 1)
UpdateValue(state, action, q, depth)
return q
```

- UCT "rollsout" by selecting actions according to the UCB1 rule

$$\arg\max_{a} \left\{ \frac{\hat{Q}(v,a)}{N(v,a)} + \sqrt{\frac{2\log N(v)}{N(v,a)}} \right\}$$

- Then updates the estimates $\hat{Q}$ in a "backup" phase up the tree.

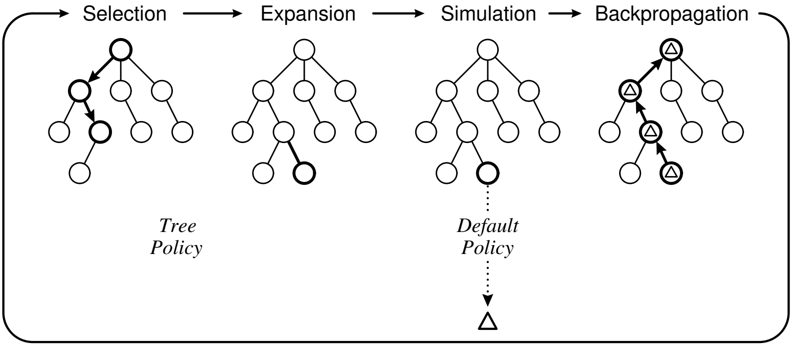
# Monte Carlo tree search

- To pick an action from state $s$, run UCT and pick the action $\arg\max_a N(v, a)$ where $v$ is the root.
- Kocsis & Szepesvári (2006) showed that at all nodes

$$\sum_a \frac{N(v, a)}{N(s)} \hat{Q}(v, a) \to V^*(s(v))$$

  - For all $v$, the prob. that UCT picks a suboptimal arm goes to 0.
  - The recursive argument relies on the fact that UCB1 has low regret and converges quickly at all nodes.
- Vanilla UCT is memory intensive, so there are many practical variants, collectively called Monte Carlo tree search (MCTS, Browne et al., 2012).

# Monte Carlo tree search




(Browne et al., 2012)

- Maintain a depth-limited subtree, select nodes using UCB1.
- Expand the tree to include excluded children, then run a simulation.
- Evaluate nodes by rolling out a default policy (not UCB1).
- Noisy evaluations are backed-up the tree.

# Monte Carlo tree search

- MCTS is fundamentally an enumerative algorithm, i.e., it must visit the whole tree.
  - But it balances exploration and exploitation and does not spend too much time on suboptimal trajectories.
- In practice, it can scale very well and discover very good actions.
  - Be careful with the scale of returns and the bandit assumption that they are in $[0, 1]$.
- It is typically applied to two-player, zero-sum, perfect information games (requires some slight modifications to the backup operator).
- Notice: MCTS can be thought of as an policy improvement operator that takes a default policy and returns a better action (i.e., policy).

# Today's talks

- AlphaZero.
- MCTS as policy optimization.
- MENTs.