Multi-Armed Bandit

What strategy do I use to pick a sequence of $a_i$?
View Multi-Armed Bandit as a Single-State MDP

\[ R(s,a_i,s) \]
\[ P(s|s,a_i)=1.0 \]
Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Pull each arm once

Algorithm:

For i ← 1 to n
{ Sum_i ← R(arm_i); N_i ← 1 }; N ← n  /* Initialization */

Loop Forever

best ← argmax \[ \frac{\text{Sum}_i}{N_i} + \frac{g(N)}{\sqrt{N_i}} \] \[ g(N) = \sqrt{2 \log (1 + N \log^2 N)} \]

pull arm a_{best} and get reward r

\text{Sum}_{best} ← \text{Sum}_{best} + r; \ N_{best} ← N_{best} + 1; \ N ← N+1
Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Algorithm:

For $i \leftarrow 1$ to $n$  
\{ $\text{Sum}_i \leftarrow \text{R}(\text{arm}_i)$; $\text{N}_i \leftarrow 1$; $\text{N} \leftarrow n$ \}  /* Initialization */

Loop Forever

$\text{best} \leftarrow \arg\max_{1 \leq i \leq n} \left[ \frac{\text{Sum}_i}{\text{N}_i} + \sqrt{\frac{2 \ln(N)}{\text{N}_i}} \right]$  \hspace{1cm} $g(N) = \sqrt{2 \log (1 + N \log^2 N)}$

pull arm $\text{a}_{\text{best}}$ and get reward $r$

$\text{Sum}_{\text{best}} \leftarrow \text{Sum}_{\text{best}} + r$; $\text{N}_{\text{best}} \leftarrow \text{N}_{\text{best}} + 1$; $\text{N} \leftarrow \text{N} + 1$

$g(N) = c \sqrt{\ln(N)}$

[c = $\sqrt{2}$]
Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why $g(N) = \sqrt{2 \log (1 + N \log^2 N)}$?

Why $g(N) = c \sqrt{\ln(N)}$?
Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why $g(N) = \sqrt{2 \log(1 + N \log^2 N)}$?

Why $g(N) = c \sqrt{\ln(N)}$?

Average reward: $\frac{\sum_{i=1}^{n} \text{Sum}_i}{N}$
Multi-Armed Bandits:
Upper Confidence Bound (UCB) Heuristic

Why $g(N) = \sqrt{2 \log (1 + N \log^2 N)}$?

Why $g(N) = c \sqrt{\ln(N)}$?

Average reward: $\frac{\sum_{i=1}^{n} \text{Sum}_i}{N}$

Average reward for a policy $\pi$: $\mu_{\pi}^N = E_{\pi} \left[ \frac{\sum_{i=1}^{n} \text{Sum}_i}{N} \right]$
Multi-Armed Bandits:
Upper Confidence Bound (UCB) Heuristic

Why \( g(N) = \sqrt{2 \log (1 + N \log^2 N)} \)?

Why \( g(N) = c \sqrt{\ln(N)} \)?

Average reward:
\[
\frac{\sum_{i=1}^{n} \text{Sum}_i}{N}
\]

Average reward for a policy \( \pi \):
\[
\mu_\pi = \mathbb{E}_\pi \left[ \frac{\sum_{i=1}^{n} \text{Sum}_i}{N} \right]
\]

Average expected reward for always picking optimal arm:
\( \mu_{\text{best}} \)
Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why \( g(N) = \sqrt{2 \log(1 + N \log^2 N)} \)?
Why \( g(N) = c \sqrt{\ln(N)} \)?

Average reward: \( \frac{\sum_{i=1}^{n} \text{Sum}_i}{N} \)

Average reward for a policy \( \pi \): \( \mu_N^{\pi} = E_\pi \left[ \frac{\sum_{i=1}^{n} \text{Sum}_i}{N} \right] \)

Average expected reward for always picking optimal arm: \( \mu^{\text{best}} \)

Regret for a policy: \( \text{regret}_N^{\pi} = \mu^{\text{best}} - \mu_N^{\pi} \)

How much exploration costs you
Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why $g(N) = \sqrt{2 \log (1 + N \log^2 N)}$?

Why $g(N) = c \sqrt{\ln(N)}$?

Known Result:
$\text{Regret}_N^\pi = \Omega(\log(N))$

Your expected regret will grow at least logarithmically with $N$.
Multi-Armed Bandits: Upper Confidence Bound (UCB) Heuristic

Why \( g(N) = \sqrt{2 \log (1 + N \log^2 N)} \)?

Why \( g(N) = c \sqrt{\ln(N)} \)?

Known Result:

\[
\text{Regret}_N^{\pi} = \Omega(\log(N))
\]

\[
\text{Regret}_N^{\text{UCB}(g(N))} = O(\log(N))
\]

UCB with these \( g(N) \) functions have regret that grows at worst logarithmically with \( N \)

Your expected regret will grow at least logarithmically with \( N \)
Monte Carlo Tree Search (MCTS)  
(Section 5.4)
Monte Carlo Tree Search (MCTS) (Section 5.4)

Application of multi-armed bandits
Timeline of Key Ideas in Game Tree Search

1948  Alan Turing  Look ahead and use an evaluation function
1950  Claude Shannon  Game tree search
1956  John McCarthy  Alpha-beta pruning
1959  Arthur Samuel  Learn evaluation function (Reinforcement learning)

...
1997: Deep Blue defeats Gary Kasparov (3½–2½)

Game tree search with alpha-beta pruning plus lots of enhancements
1997: **Deep** Blue defeats Gary Kasparov (3½–2½)
(Not “deep” as in “deep learning”)
(Deep as in its ancestor, Deep Thought)

Game tree search with alpha-beta pruning
plus lots of enhancements
Go?

Branching factor is in the 100s

Evaluation function is difficult because payoff may be very far away

Need new ideas
<table>
<thead>
<tr>
<th>Year</th>
<th>Author(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>Bruce Ballard</td>
<td>Lookahead for probabilistic moves</td>
</tr>
<tr>
<td>1987</td>
<td>Bruce Abramson</td>
<td>Evaluation by expected outcome (repeated simulation)</td>
</tr>
<tr>
<td>1992</td>
<td>Gerald Tesauro</td>
<td>TD-Gammon (reinforcement learning, self-play)</td>
</tr>
<tr>
<td>1992</td>
<td>Bernd Brugmann</td>
<td>Monte Carlo Go (simulated annealing)</td>
</tr>
<tr>
<td>1999</td>
<td>U of Alberta</td>
<td>Simulation in Poker</td>
</tr>
<tr>
<td>1999</td>
<td>Matt Ginsberg</td>
<td>Simulation in Bridge</td>
</tr>
<tr>
<td>2002</td>
<td>Brian Sheppard</td>
<td>Simulation in Scrabble</td>
</tr>
<tr>
<td>2006</td>
<td>Levente Kocsis and Csaba Szepesvár</td>
<td>Multi-armed bandits for Monte-Carlo tree search</td>
</tr>
</tbody>
</table>
2016: AlphaGo defeats Lee Sedol (4-1)
Key ideas of Monte Carlo Tree Search:

1. View move selection as a multi-armed bandit problem
Multi-Armed Bandit

$M_1, M_2, M_3, M_4, \ldots, M_n$

$a_1, a_2, a_3, a_4, \ldots, a_n$

$R_1, R_2, R_3, R_4, \ldots, R_n$
Multi-Armed Bandit for Game Tree Search

\[ a_1, a_2, a_3, a_4, \ldots, a_n \]

\[ M_1, M_2, M_3, M_4, \ldots, M_n \]

\[ R_1, R_2, R_3, R_4, \ldots, R_n \]
Multi-Armed Bandit for Game Tree Search

What move should I try?
Key ideas of Monte Carlo Tree Search:

1. View move selection as a multi-armed bandit problem
2. Evaluate moves by simulating games
Multi-Armed Bandit for Game Tree Search

What move should I try on each simulated game?
Monte-Carlo Tree Search (MCTS) Terms

• Leaf node: A state in the game tree that has successors for which no games have been simulated
• Terminal node: End of game state
• Playout/rollout: Simulating a game from a leaf node to a terminal node
Three Steps in MCTS

• Selection: Make move choices until a leaf node $S$ is reached
Three Steps in MCTS

(a) Selection
Three Steps in MCTS

(a) Selection
Three Steps in MCTS

(a) Selection
Three Steps in MCTS

(a) Selection
Three Steps in MCTS

• Selection: Make move choices until a leaf node $S$ is reached
• Expansion: Create a new successor state $S'$ for an untried action
  Simulation: Play a game until you reach a terminal node
Three Steps in MCTS

(a) Selection
(b) Expansion and Simulation
black wins
Three Steps in MCTS

• Selection: Make move choices until a leaf node S is reached
• Expansion: Create a new successor state S’ for an untried action
  Simulation: Play a game until you reach a terminal node
• Backpropagation: Update game statistics for the path from S’ up to the root
Three Steps in MCTS

(a) Selection  
(b) Expansion and Simulation  
(c) Backpropagation
MCTS(state):

while TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACKPROPAGATE(result, child)

return argmax \#playouts(apply(a, state))_{a \in A}

Which move gives the game state with most playouts
Three Steps in MCTS

How do we pick moves?

(a) Selection
(b) Expansion and Simulation
(c) Backpropagation
Remember This?
(UCB)

Algorithm:

Pull each arm once
For i ← 1 to n { Sum_i ← R(arm_i); N_i ← 1 }; N ← n  /* Initialization */
Loop Forever

best ← \arg\max_{1 \leq i \leq n} \left[ \frac{\text{Sum}_i}{N_i} + c \sqrt{\frac{\ln N}{N_i}} \right]

pull arm a_{best} and get reward r

Sum_{best} ← Sum_{best} + r; N_{best} ← N_{best} + 1; N ← N+1
Picking a Move During Selection and Expansion (UCT – Upper Confidence bound applied to Trees)

\[
\text{Sum}_i = \# \text{ of wins} \\
N_i = \# \text{ of times } i \text{ was tried} \\
N = \# \text{ of simulations thus far (N(parent(i)))}
\]

\[
\text{best} \leftarrow \text{argmax}_{1 \leq i \leq n} \left[ \frac{\text{Sum}_i}{N_i} + c \sqrt{\frac{\ln N}{N_i}} \right]
\]

Lets you control how much exploration
Three Steps in MCTS

(a) Selection
(b) Expansion and Simulation
(c) Backpropagation

How do we pick moves?
Picking a Move During Simulation

- Light playout: Pick uniformly at random

- Heavy playout: Make a biased selection
  - Simulation statistics
  - Game knowledge

Trade off: Slower run time vs missing a move
Benefits

• Doesn’t use an evaluation function!
• Time is linear in depth
• Handles large number of actions
• Let’s you make a move when a timer goes off (to manage time) (‘’anytime algorithm’’)

AlphaGo / AlphaZero

• Truncated playouts and used (learned) evaluation function
• UCB with additional term for (learned) probability of win