Monte-Carlo Planning: Basic Principles and Recent Progress

Dan Weld – UW CSE 573
October 2012

Most slides by
Alan Fern
EECS, Oregon State University
A few from me, Dan Klein, Luke Zettlemoyer, etc

Logistics 1 – HW 1

• Consistency & admissibility
• Correct & resubmit by Mon 10/22 for 50% of missed points

Logistics 2

• HW2 – due tomorrow evening
• HW3 – due Mon10/29
  • Value iteration
  • Understand terms in Bellman eqn
  • Q-learning
  • Function approximation & state abstraction

Logistics 3

Projects
• Teams (~3 people)
• Ideas

Outline

• Recap: Markov Decision Processes
• What is Monte-Carlo Planning?
• Uniform Monte-Carlo
  • Single State Case (PAC Bandit)
  • Policy rollout
  • Sparse Sampling
• Adaptive Monte-Carlo
  • Single State Case (UCB Bandit)
  • UCT Monte-Carlo Tree Search
• Reinforcement Learning

Stochastic/Probabilistic Planning: Markov Decision Process (MDP) Model

We will model the world as an MDP.
An MDP has four components: $S$, $A$, $P_R$, $P_T$:

- finite state set $S$
- finite action set $A$
- Transition distribution $P_T(s' \mid s, a)$
  - Probability of going to state $s'$ after taking action $a$ in state $s$
  - First-order Markov model
- Bounded reward distribution $P_R(r \mid s, a)$
  - Probability of receiving immediate reward $r$ after executing $a$ in $s$
  - First-order Markov model

**First-Order Markovian dynamics** (history independence)
- Next state only depends on current state and current action

**First-Order Markovian reward process**
- Reward only depends on current state and action

**Recap: Defining MDPs**

- Policy, $\pi$
  - Function that chooses an action for each state
- Value function of policy
  - Aka Utility
  - Sum of discounted rewards from following policy
- Objective?
  - Find policy which maximizes expected utility, $V(s)$

**Value Function of a Policy**

- We consider finite-horizon discounted reward, discount factor $0 \leq \beta < 1$
- $V_\pi(s,h)$ denotes expected $h$-horizon discounted total reward of policy $\pi$ at state $s$
- Each run of $\pi$ for $h$ steps produces a random reward sequence: $R_1, R_2, \ldots, R_h$
- $V_\pi(s,h)$ is the expected discounted sum of this sequence

$$V_\pi(s,h) = E \left[ \sum_{t=0}^{h} \beta^t R_t \mid \pi, s \right]$$
- Optimal policy $\pi^*$ is policy that achieves maximum value across all states

**Relation to Infinite Horizon Setting**

- Often value function $V_\pi(s)$ is defined over infinite horizons for a discount factor $0 \leq \beta < 1$

$$V_\pi(s) = E \left[ \sum_{t=0}^{\infty} \beta^t R_t \mid \pi, s \right]$$

- It is easy to show that difference between $V_\pi(s,h)$ and $V_\pi(s)$ shrinks exponentially fast as $h$ grows

$$\left| V_\pi(s) - V_\pi(s,h) \right| \leq \left( \frac{R_{\text{max}}}{1-\beta} \right)^h$$
- $h$-horizon results apply to infinite horizon setting
**Bellman Equations for MDPs**

\[ V^*(s) = \max_{a \in A(s)} \sum_{s' \in S} P_{r}(s'|s, a) \{ R(s, a, s') + \gamma V^*(s') \} \]

\[ V^*(s) = \max_a Q^*(s, a) \]

**Computing the Best Policy**

- Optimal policy maximizes value at each state
- Optimal policies guaranteed to exist [Howard, 1960]
- When state and action spaces are small and MDP is known we find optimal policy in polynomial time
  - With value iteration
  - Or policy iteration
- Both use…?

**Bellman Backup**

\[ V_{t+1}(s) = \max_{a \in A(s)} \sum_{s' \in S} P_{r}(s'|s, a) \{ R(s, a, s') + \gamma V_t(s') \} \]

**Computing the Best Policy**

What if…

- Space is exponentially large?
- MDP transition & reward models are unknown?

**Large Worlds: Model-Based Approach**

1. Define a language for compactly describing MDP model, for example:
   - Dynamic Bayesian Networks
   - Probabilistic STRIPS/PDDL
2. Design a planning algorithm for that language

**Problem**: more often than not, the selected language is inadequate for a particular problem, e.g.

- Problem size blows up
- Fundamental representational shortcoming

**Large Worlds: Monte-Carlo Approach**

- 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
  - Fire & Emergency Response
  - Klondike Solitaire
Large Worlds: Monte-Carlo Approach

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
- 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.

MDP: Simulation-Based Representation

- A simulation-based representation gives: S, A, R, T:
  - finite state set S (generally very large)
  - finite action set A
  - Stochastic, real-valued, bounded reward function \( R(s,a) = r \)
    - Stochastically returns a reward \( r \) given input \( s \) and \( a \)
    - Can be implemented in arbitrary programming language
  - 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
    - \( T \) can be implemented in an arbitrary programming language

Slot Machines as MDP?

Outline

- Preliminaries: Markov Decision Processes
- What is Monte-Carlo Planning?
- Uniform Monte-Carlo
  - Single State Case (Uniform Bandit)
  - Policy rollout
  - Sparse Sampling
- Adaptive Monte-Carlo
  - Single State Case (UCB Bandit)
  - UCT Monte-Carlo Tree Search

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 \( R(s,a) \)
**PAC Bandit Objective**

**Probably Approximately Correct (PAC)**

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

\[ R(s, a_1) \quad R(s, a_2) \quad \ldots \quad R(s, a_k) \]

**Multi-Armed Bandit Problem**

**Aside: Additive Chernoff Bound**

- Let \(R\) be a random variable with maximum absolute value \(Z\). An let \(r_i\) (for \(i=1,\ldots,w\)) 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 \(E[R]\)

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

Equivalently:

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

\[
|E[R] - \frac{1}{w} \sum_{i=1}^{w} r_i| \leq Z \sqrt{\frac{\varepsilon^2}{2w}} \ln \frac{1}{\delta}
\]

**UniformBandit PAC Bound**

With a bit of algebra and Chernoff bound we get:

If \(w \geq \left( \frac{R_{\text{max}}}{\varepsilon^2} \right)^{\ln \frac{1}{\delta}}\) for all arms simultaneously

\[
E[R(s, a_i)] - \frac{1}{w} \sum_{i=1}^{w} r_i \leq \varepsilon
\]

with probability at least \(1-\delta\)

- That is, estimates of all actions are \(\varepsilon\)-accurate with probability at least \(1-\delta\)
- Thus selecting estimate with highest value is approximately optimal with high probability, or PAC

**UniformBandit Algorithm**

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

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

\[ s \quad a_1 \quad a_2 \quad \ldots \quad a_k \]

\[ r_{11} \quad r_{21} \quad \ldots \quad r_{kw} \]

How large must \(w\) be to provide a PAC guarantee?

**# Simulator Calls for UniformBandit**

Total simulator calls for PAC:

\[ k \cdot w = O\left( \frac{k}{\varepsilon^2 \ln \frac{1}{\delta}} \right) \]

* Can get rid of \(\ln(k)\) term with more complex algorithm [Even-Dar et. al., 2002].
Outline

- Preliminaries: Markov Decision Processes
- What is Monte-Carlo Planning?
- Non-Adaptive Monte-Carlo
  - Single State Case (PAC Bandit)
  - Policy rollout
  - Sparse Sampling
- Adaptive Monte-Carlo
  - Single State Case (UCB Bandit)
  - UCT Monte-Carlo Tree Search

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?

World Simulator + Base Policy

Policy Improvement Theorem

- The h-horizon Q-function $Q_\pi(s,a,h)$ is defined as:
  the expected total discounted reward of starting in state $s$, taking
  action $a$, and then following policy $\pi$ for $h$-1 steps.
- Define: $\pi'(s) = \arg \max_a Q_\pi(s,a,h)$
- Theorem [Howard, 1960]: For any non-optimal policy $\pi$ the
  policy $\pi'$ a strict improvement over $\pi$.
- Computing $\pi'$ amounts to finding the action that maximizes
  the Q-function
- Can we use the bandit idea to solve this?

Policy Improvement via Bandits

- Idea: define a stochastic function $\text{Sim}Q(s,a,\pi,h)$ that we
  can implement and whose expected value is $Q_\pi(s,a,h)$
- Use Bandit algorithm to PAC select improved action

How to implement $\text{Sim}Q$?

Policy Improvement via Bandits

- Simply simulate taking $a$ in $s$ and following policy for $h$-1
  steps, returning discounted sum of rewards
- Expected value of $\text{Sim}Q(s,a,\pi,h)$ is $Q_\pi(s,a,h)$
Policy Rollout Algorithm

1. For each $a_i$, run $\text{SimQ}(s, a_i, \pi, h)$ $w$ times
2. Return action with best average of $\text{SimQ}$ results

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

Multi-Stage Rollout

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

Policy Rollout: # of Simulator Calls

- For each action, $w$ calls to $\text{SimQ}$, each using $h$ sim calls
- Total of $khw$ calls to the simulator

Policy Rollout: # of Simulator Calls

- For each action, $w$ calls to $\text{SimQ}$, each using $h$ sim calls
- Total of $khw$ calls to the simulator

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]

<table>
<thead>
<tr>
<th>Player</th>
<th>Success Rate</th>
<th>Time/Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Expert</td>
<td>36.6%</td>
<td>20 min</td>
</tr>
<tr>
<td>(naïve) Base Policy</td>
<td>13.05%</td>
<td>0.021 sec</td>
</tr>
</tbody>
</table>

Example: Rollout for Thoughtful Solitaire

[Yan et al. NIPS’04]

<table>
<thead>
<tr>
<th>Player</th>
<th>Success Rate</th>
<th>Time/Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Expert</td>
<td>36.6%</td>
<td>20 min</td>
</tr>
<tr>
<td>(naïve) Base Policy</td>
<td>13.05%</td>
<td>0.021 sec</td>
</tr>
<tr>
<td>1 rollout</td>
<td>31.20%</td>
<td>0.67 sec</td>
</tr>
</tbody>
</table>
Example: Rollout for Thoughtful Solitaire
[Yan et al. NIPS’04]

<table>
<thead>
<tr>
<th>Player</th>
<th>Success Rate</th>
<th>Time/Game</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Expert</td>
<td>36.6%</td>
<td>20 min</td>
</tr>
<tr>
<td>(naïve) Base</td>
<td>13.05%</td>
<td>0.021 sec</td>
</tr>
<tr>
<td>1 rollout</td>
<td>31.20%</td>
<td>0.67 sec</td>
</tr>
<tr>
<td>2 rollout</td>
<td>47.6%</td>
<td>7.13 sec</td>
</tr>
</tbody>
</table>

Deeper rollout can pay off, but is expensive

**Outline**

- Preliminaries: Markov Decision Processes
- What is Monte-Carlo Planning?

  - Uniform Monte-Carlo
    - Single State Case (UniformBandit)
    - Policy rollout
    - Sparse Sampling
  - Adaptive Monte-Carlo
    - Single State Case (UCB Bandit)
    - UCT Monte-Carlo Tree Search

**MDP Basics**

- Let $V^*(s,h)$ be the optimal value function of MDP
- Define $Q^*(s,a,h) = E[R(s,a) + V^*(T(s,a),h-1)]$
  - Optimal $h$-horizon value of action $a$ at state $s$.
  - $R(s,a)$ and $T(s,a)$ return random reward and next state.
- Optimal Policy: $\pi^*(x) = \text{argmax}_a Q^*(x,a,h)$
- What if we knew $V^*$?
  - Can apply bandit algorithm to select action that approximately maximizes $Q^*(s,a,h)$

**Sparse Sampling**

- Rollout does not guarantee optimality or near optimality
- Can we develop simulation-based methods that give us near optimal policies?
  - Using computation that doesn’t depend on number of states!
- In deterministic games and problems it is common to build a look-ahead tree at a state to determine best action
  - Can we generalize this to general MDPs?
  - **Sparse Sampling** is one such algorithm
    - Strong theoretical guarantees of near optimality

**Bandit Approach Assuming $V^*$**

- $\text{SimQ}^*(s,a,h) = E[R(s,a) + V^*(T(s,a),h-1)]$
  - $R(s,a)$ and $T(s,a)$ return random reward and next state.
  - Expected value of $\text{SimQ}^*(s,a,h)$ is $Q^*(s,a,h)$
    - Use UniformBandit to select approximately optimal action
But we don’t know \( V^* \)

- To compute \( \text{SimQ}^*(s,a,h) \) need \( V^*(s',h-1) \) for any \( s' \)
- Use recursive identity (Bellman’s equation):
  \[ V^*(s,h-1) = \max_a Q^*(s,a,h-1) \]
- Idea: Can recursively estimate \( V^*(s,h-1) \) by running \( h-1 \) horizon bandit based on \( \text{SimQ}^* \)
- Base Case: \( V^*(s,0) = 0 \), for all \( s \)

Sparse Sampling [Kearns et. al. 2002]

This recursive UniformBandit is called Sparse Sampling

Return value estimate \( V^*(s,h) \) of state \( s \) and estimated optimal action \( a^* \)

\[ \text{SparseSampleTree}(s,h,w) \]

For each action \( a \) in \( s \)
- \( Q^*(s,a,h) = 0 \)

For \( i = 1 \) to \( w \)
  - Simulate taking \( a \) in \( s \) resulting in \( s' \), and reward \( r \); 
  - \( [V^*(s,h),a^*] = \text{SparseSample}(s,h,1,w) \)
  - \( Q^*(s,a,h) = Q^*(s,a,h) + r + V^*(s',h) \)
  - \( Q^*(s,a,h) = Q^*(s,a,h) / w \) : estimate of \( Q^*(s,a,h) \)

\( V^*(s,h) = \max_a Q^*(s,a,h) \) : estimate of \( V^*(s,h) \)

\( a^* = \text{argmax}_a Q^*(s,a,h) \)

Return \([V^*(s,h), a^*]\)

Outline

- Preliminaries: Markov Decision Processes
- What is Monte-Carlo Planning?
- Uniform Monte-Carlo
  - Single State Case (UniformBandit)
  - Policy rollout
  - Sparse Sampling
- Adaptive Monte-Carlo
  - Single State Case (UCB Bandit)
  - UCT Monte-Carlo Tree Search

Sparse Sampling

- For a given desired accuracy, how large should sampling width and depth be?
  - Answered: [Kearns et. al., 2002]
- Good news: can achieve near optimality for value of \( w \) independent of state-space size!
  - First near-optimal general MDP planning algorithm whose runtime didn’t depend on size of state-space
- Bad news: the theoretical values are typically still intractably large—also exponential in \( h \)
- In practice: use small \( h \) and use heuristic at leaves (similar to minimax game-tree search)

\( \# \) of Simulator Calls

- Can view as a tree with root \( s \)
- Each state generates \( kw \) new states (\( w \) states for each of \( k \) bandits)
- \# total \# of states in tree \((kw)^h\)
- How large must \( w \) be?
Uniform vs. Adaptive Bandits

- Sparse sampling wastes time on bad parts of tree
  - 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:
  $$a^* = \arg \max_a Q(a) + \sqrt{\frac{2 \ln n}{n(a)}}$$
  Assumes payoffs in $[0,1]$

Value Term: favors actions that looked good historically

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

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 $\Delta_a$ is regret of arm $a$

UCB for Multi-State MDPs

- UCB-Based Policy Rollout:
  - Use UCB to select actions instead of uniform
- UCB-Based Sparse Sampling
  - Use UCB to make sampling decisions at internal tree nodes

UCB-based Sparse Sampling [Chang et. al. 2005]

- Use UCB instead of Uniform to direct sampling at each state
- Non-uniform allocation

- But each $q_i$ sample requires waiting for an entire recursive level tree search
- Better but still very expensive!
Outline

- Preliminaries: Markov Decision Processes
- What is Monte-Carlo Planning?
- Uniform Monte-Carlo
  - Single State Case (UniformBandit)
  - Policy rollout
  - Sparse Sampling
- Adaptive Monte-Carlo
  - Single State Case (UCB Bandit)
  - UCT Monte-Carlo Tree Search

UCT Algorithm [Kocsis & Szepesvari, 2006]

- Instance of Monte-Carlo Tree Search
  - Applies principle of UCB
  - Some nice theoretical properties
  - Much better anytime behavior than sparse sampling
  - 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.

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.
**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

\[
\pi_{UCT}(s) = \arg\max_a Q(s,a) + c \left( \frac{\ln n(s)}{n(s,a)} \right)
\]

*Theoretical constant that must be selected empirically in practice*

**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)

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.)

Computer GO Server: 1800 ELO → 2600 ELO

Other Successes

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

- List is growing

- Usually extend UCT is some ways

Some Improvements

- Use domain knowledge to handcraft a more intelligent default policy than random
  - E.g. don’t choose obviously stupid actions
- Learn a heuristic function to evaluate positions
  - Use the heuristic function to initialize leaf nodes (otherwise initialized to zero)

Summary

- When you have a tough planning problem and a simulator
  - Try Monte-Carlo planning
- Basic principles derive from the multi-arm bandit
- Policy Rollout is a great way to exploit existing policies and make them better
- If a good heuristic exists, then shallow sparse sampling can give good gains
- UCT is often quite effective especially when combined with domain knowledge