AI Planning for Autonomy

1. Monte Carlo Tree Search

Balancing Exploitation and Exploration

Tim Miller

THE UNIVERSITY OF MELBOURNE

Winter Term 2017

Agenda

Relevant Reading

- Chapters 2 and 5 of *Reinforcement Learning: An Introduction. Second edition* by Richard S. Sutton and Andrew G. Barto
  A *draft* is available at: `http://ufal.mff.cuni.cz/~straka/courses/npfl114/2016/sutton-bookdraft2016sep.pdf`
- *A Survey of Monte Carlo Tree Search Methods* by Browne et al. *IEEE Transactions on Computational Intelligence and AI in Games*, 2012
  → Good "entry level" resource, with lots of pointers to seminal papers
- Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems by S. Bubeck and N. Cesa–Bianchi, 2012
  → All you want to know about regret analysis and multi-armed bandits

Agenda

Offline Planning & Online Planning over MDPs

We saw value iteration and policy iteration in the previous lecture. These are *offline* planning methods, in that we solve the problem offline for all possible states, and then use the solution (a policy) online. These offline planning policies π are typically a value function $V(s)$, such that:

- We can define policies that work from any state in a convenient manner,
- Yet the state space S is usually *far* too big to determine $V(s)$ exactly.
- There are methods to approximate the MDP by reducing the dimensionality of S, but we will not discuss these in this subject.

Online Planning policies are defined procedurally. That is, actions are selected online at each state, with the calculation of which action to select being done during execution.

- For each state s visited, many policies π are *evaluated* (partially)
- The quality of each π iis approximated by averaging the expected reward of trajectories over S obtained by repeated simulations of $r(s, a, s')$.
- The chosen policy $\hat{\pi}$ is selected and the action $\hat{\pi}(s)$ executed.

The question is: how to we do the repeated simuations. Monte Carlo methods are by far the most widely-used approach.

Agenda

Monte Carlo

Monte Carlo is an area within Monaco (small principality on the French reriviera), which is best known for its extravagent casinos. As gambling and casinos are largely associated with chance, methods for solving MDPs online are often called *Monte Carlo* methods, because they use *randomness* to search the action space.

Figure : By I, Katonams, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2480853

Monte Carlo Tree Search – Overview

The algorithm is online, which means the search is done while the agent is executing, rather than beforehand. Thus, MCTS is invoked every time it visits a new state s. Fundamental features:

1. The Q-value for each action a at state s $Q(s, a)$ is approximated using *random simulation*.
2. These estimates of $Q(s, a)$ are used to drive a *best-first strategy*; thus, $Q(s, a)$ is being updated while it is also be used as an heuristic in the search.
3. A *search tree* is built incrementally
4. The search terminates when some pre-defined computational budget is used up, such as a time limit or a number of expanded nodes.
   Therefore, it is an *anytime* algorithm, as it can be terminated at any time and still give an answer.
5. The best performing action a at s is returned; that is $argmax_{a \in A} Q(s, a)$.

- $\rightarrow$ This is complete if there are *no* dead–ends.
- $\rightarrow$ This is optimal if if the computational budget, and is perfect if an entire search can be performed (which is unusual).

Monte-Carlo Tree Search: Sketch of the Algorithm

The Framework: Monte Carlo Tree Search (MCTS)
The evaluated states are stored in a search tree. The set of evaluated states is incrementally built be iterating over the following four steps:

- *Select*: Given a *tree policy*, select a single node in the tree to assess.
- *Expand*: Expand this node by applying one available action from the node.
- *Simulation*: From the expanded node, perform a complete random simulation to a leaf node. This therefore assumes that the search tree is finite (but version for infinitely large trees exist).
- *Backpropagate*: Finally, the value of the node is *backpropagated* to the root node, updating the value of each ancestor node on the way.

→ Mentioned in "bits and pieces" by many works, one of the earliest comprehensive discussions of MCTS is this one:

M. Ginsberg: *GIB: Steps Toward an Expert-Level Bridge-Playing Program*, 1999
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.3210&rep=rep1&type=pdf`

Monte Carlo Tree Search: Sketch of the Algorithm (cont'd)

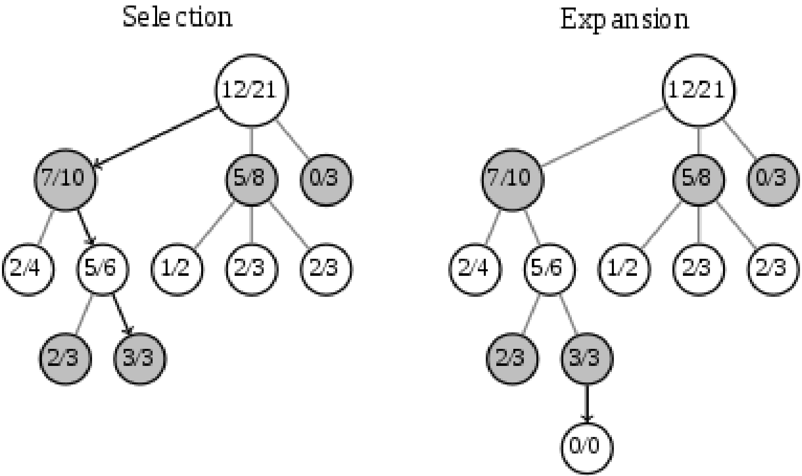
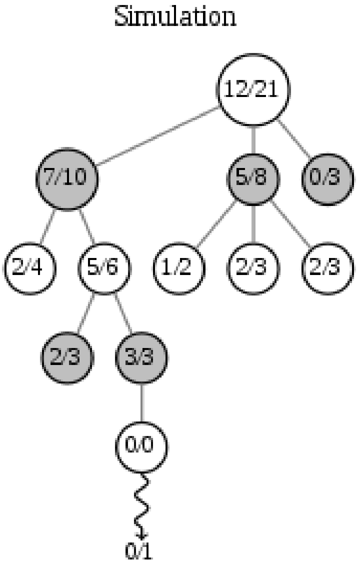


Image courtesy of Wikipedia: `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`

Monte Carlo Tree Search: Sketch of the Algorithm (cont'd)



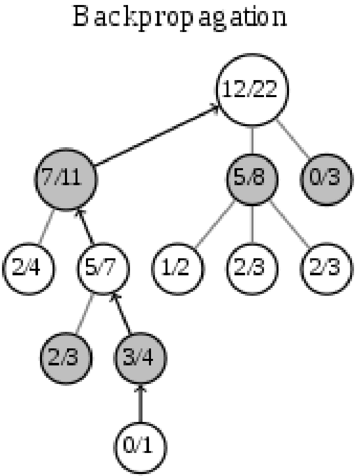


Image courtesy of Wikipedia: `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`

Monte Carlo Tree Search: Algorithm

Input: MDP M, with initial state s_0 set to current state s.

function MctsSearch $M = \langle S, s_0, A, P_a(s, s'), R(s) \rangle$ **returns** action a
  $root$:= CreateRootNode(s_0);
  while within computational budget **do**
    $best$:= TreePolicy($root$);
    Δ := DefaultPolicy($best$);
    Backup($best$, Δ);
  return $\text{argmax}_a\, Q(a, s_0)$;

→ **Nodes** store pair s, a, pointer to parent and $Q(a, s)$

→ **Node Expansion**

- → Simulate $P_a(s, s')$ to obtain *successor* s'. That is, we essentially execute action a from the MDP model in state s, choosing state s' with probability $P_a(s, s')$.
- → Create nodes for pairs (s', b), $b \in A$ and executable on s'.

Monte Carlo Tree Search: Algorithm

Input: MDP M, with initial state s_0 set to current state s.

function MCTSSEARCH $M = \langle S, s_0, A, P_a(s, s'), R(s) \rangle$ **returns** action a
  $root$:= CREATEROOTNODE(s_0);
  while within computational budget **do**
    $best$:= TREEPOLICY($root$);
    Δ := DEFAULTPOLICY($best$);
    BACKUP($best$, Δ);
  return $\text{argmax}_a\ Q(a, s_0)$;

TREEPOLICY($root$)

- → Select *recursively* the *most promising* node (s, a) to expand.
- → Check if the generated nodes (s', b) are already in tree.
- → If not in the search tree, *add* these nodes to the tree.
- → Return last node as $best$.

Important: $P_a(s, s')$ is *stochastic*, so several visits (in theory an infinite number) may be necessary to generate all successors.

Monte Carlo Tree Search: Algorithm

Input: MDP M, with initial state s_0 set to current state s.

function MCTSSEARCH $M = \langle S, s_0, A, P_a(s, s'), R(s) \rangle$ **returns** action a
  $root$:= CREATEROOTNODE(s_0);
  while within computational budget **do**
    $best$:= TREEPOLICY($root$);
    Δ := DEFAULTPOLICY($best$);
    BACKUP($best$, Δ);
  return $\text{argmax}_a \, Q(a, s_0)$;

DEFAULTPOLICY($best$)

- → Arbitrary policy $\pi(s')$ used to evaluate below node (s, a).
- → Δ is the *maximal reward* attained by $\pi(s')$.
- → In some applications $\pi(s')$ is hand-coded.
- → Domain independent version of $\pi(s')$: random walk, simulated annealing, ...

Intuition: Think of DEFAULTPOLICY as a *heuristic*.

Monte Carlo Tree Search: Algorithm

Input: MDP M, with initial state s_0 set to current state s.

function $\textsc{MctsSearch}$ $M = \langle S, s_0, A, P_a(s,s'), R(s)\rangle$ **returns** action a
  $root := \textsc{CreateRootNode}(s_0)$;
  while within computational budget **do**
    $best := \textsc{TreePolicy}(root)$;
    $\Delta := \textsc{DefaultPolicy}(best)$;
    $\textsc{Backup}(best, \Delta)$;
  return $\operatorname{argmax}_a Q(a, s_0)$;

$\textsc{Backup}(best,\Delta)$

→ Δ is backpropagated from $best$ to their parents recursively.

→ A *discount factor* can be used, as in value iteration, policy iteration, etc.

Agenda

Informed Search

There is one key question question that we need to answer:

How do we select the next node to expand?

It turns out that this selection makes a big difference on the performance of MCTS.

Multi-Armed Bandit: Informal Definition

The selection of nodes can be considered an instance of the *Multi-armed bandit* problem. This problem is defined as follows:

> *Imagine that you have N number of slot machines (or poker machines in Australia), which are sometimes called one-armed bandits. Over time, each bandit pays a random reward from an unknown probability distrbution. Some bandits pay higher rewards than others. The goal is to maximize the sum of the rewards of a sequence of lever pulls of the machine.*

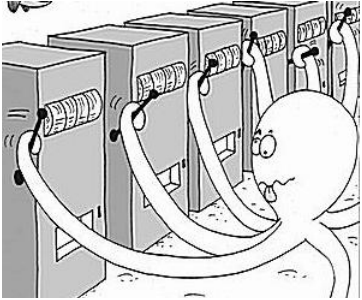


Image courtest of Mathworks blog: `https://blogs.mathworks.com/loren/2016/10/10/multi-armed-bandit-problem-and-exploration-vs-exploitation-trade-off/`

Multi-Armed Bandit: Formal Definition

An N-armed bandit is defined by a set of *random variables* $X_{i,k}$ where

- $1 \leq i \leq N$, such that i is the *arm* of the bandit; and
- k the index of the *play* of arm i.

Successive plays $X_{i,1}, X_{j,2}, X_{k,3} \ldots$ are assumed to be independently distributed according to an *unknown* law. That is, we do not know the probability distributions of the random variables.

Intuition: actions a applicable on s are the "arms of the bandit", and $Q(a, s)$ corresponds to the random variables $X_{i,n}$.

Flat Monte Carlo (FMC) a.k.a Uniform Sampling

Given that we do not know the distributions, a simple strategy is simply to select the arm given a uniform distribution; that is, select each arm with the same probability. This is just uniform sampling.

Then, the Q-value for an action a in a given state s can be approximated using the following formula:

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{t=1}^{N(s)} \mathbb{I}_t(s,a) r_t$$

- $N(s,a)$ is the number of times a executed in s.
- $N(s)$ is the number of times s is visited.
- r_t is the *reward* obtained by the t-th simulation from s.
- $\mathbb{I}_t(s,a)$ is 1 if a was selected on the t-th simulation from s, and is 0 otherwise

→ FMC suffices to achieve *world champion level* play on Bridge (Ginsberg, 01) and Scrabble (Sheppard, 02).

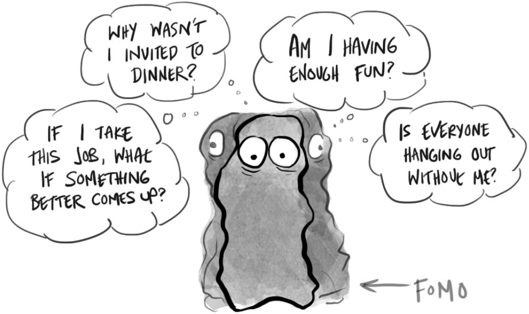
But what is the issue? Sampling Time is wasted equally in all actions using the uniform distribution. Why not focus also on the *most promising parts* of the tree given the rewards we have received so far.

Exploration vs. Exploitation

What we want is to play only the good actions; so just keep playing the actions that have given us the best reward so far. However, our selection is randomised, so what if we just haven't sampled the best action enough times? Thus, we want strategies that *exploit* what we think are the best actions so far, but still *explore* other actions.

But how much should we exploit and how much should we explore? This is known as the *exploration vs. exploitation dilemma*.

It is driven by the *The Fear of Missing Out* (FOMO).



The Fear Of Missing Out

We seek policies π that *minimise regret*.

(Pseudo)–Regret

$$\mathcal{R}_{N(s),b} = Q(\pi^*(s), s)N(s) - \mathbb{E}[\sum_{t}^{N(s)} Q(b, s)\mathbb{I}_t(s, b)]$$

$Q(\pi^*(s), s)$ is the Q-value for the (unknown) optimal policy $\pi^*(s)$,

$N(s)$ is the number of visits to state s,

$\mathbb{I}_i(s, a)$ is 1 if a was selected on the i-th visit from s, and 0 otherwise,

Important: $\mathbb{E}[\sum_t^{N(s)} Q(b, s)\mathbb{I}_t(s, b)] > 0$ for every b.

Informally: If I play arm b, my regret is the *best possible expected reward* minus the *expected reward of playing* b. If I play arm a (the best arm), my regret is 0. *Regret* is thus the *expected loss* due to not doing the best action.

$\rightarrow$ In multi-armed bandit algorithms, exploration is *literally* driven by FOMO.

Solutions that aim to minimise regret

ϵ-**greedy**: ϵ is a number in [0,1]. Each time we need to choose an arm, we choose a random arm with probability ϵ, and choose the arm with max $Q(s,a)$ with probability $1-\epsilon$. Typically, values of ϵ around 0.05-0.1 work well.

ϵ-**decreasing**: The same as ϵ-greedy, ϵ decreases over time. A parameter α between [0,1] specifies the *decay*, such that $\epsilon := \epsilon.\alpha$ after each action is chosen.

Softmax: This is *probability matching strategy*, which means that the probability of each action being chosen is dependent on its Q-value so far. Formally:

$$\frac{e^{Q(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q(s,b)/\tau}}$$

in which τ is the *temperature*, a positive number that dictates how much of an influence the past data has on the decision.

Upper Confidence Bounds (UCB1)

A highly effective (especially in terms of MCTS) multi-armed bandit strategy is the *Upper Confidence Bounds* (UCB1) strategy.

UCB1 policy $\pi(s)$

$$\pi(s) := \underset{a \in A(s)}{\operatorname{argmax}}\ Q(a,s) + \sqrt{\frac{2 \ln N(s)}{N(a,s)}}$$

$Q(a,s)$ is the estimated Q-value.

$N(s)$ is the number of times s has been visited.

$N(s,a)$ is the number of times times a has been executed in s.

$\rightarrow$The left–hand side encourages exploitation: the Q-value is high for actions that have had a high reward.
$\rightarrow$The right–hand side encourages exploration: it is high for actions that have been explored less.

Agenda

Upper Confidence Trees (UCT)

UCT = MCTS + UCB1

Kocsis & Szepesvári were the first to treat the selection of nodes to expand in MCTS as a multi-armed bandit problem.

UCT exploration policy

$$\pi(s) := \underset{a \in A(s)}{\operatorname{argmax}}\, Q(a,s) + 2C_p\sqrt{\frac{2\ln N(s)}{N(s,a)}}$$

$C_p > 0$ is the exploration constant, which determines can be increased to encourage more exploration, and decreased to encourage less exploration. Ties are broken randomly.

$\rightarrow$ if $Q(a,s) \in [0,1]$ **and** $C_p = \frac{1}{\sqrt{2}}$ **then** in two-player adversarial games, UCT converges to the well-known Minimax algorithm (if you don't know what Minimax is, ignore this for now and we'll mention it later in the subject).

Applications of MCTS with UCB tree policies

Games:

- Go: MoGo (2006), Fuego (2009), ..., Alpha Go(2010–2016)
- Board Games: Havannah, Y, Cataan, Othello, Arimaa...
- Video Games: Atari 2600

Not Games:

- Computer Security: Attack tree generation & Penetration testing
- Deep Learning: Automated "performance tuning" of Neural Nets and Feature Selection
- Operations Research: Optimising bus schedules, energy stock management...

Why does it work so well (sometimes)?

It addresses exploitation vs. exploration comprehensively.

- UCT is *systematic*:
  - Policy evaluation is *exhaustive* up to a certain depth.
  - Exploration aims at *minimising regret* (or FOMO).

Watch it playing MARIO BROS.

Where it does not do so well..: Atari 2600 game FREEWAY. It fails here because the character does not receive a reward until it reaches the other side of the road, so UCT has no feedback to go on.

Agenda

Summary

- Monte Carlo Tree Search (MCTS) is an anytime search algorithm, especially good for stochastic domains, such as MDPs.
  - Smart selection strategies are *crucial* for good performance.
- Upper Confidence Bounds (UCB1) for Multi-Armed Bandits makes a good selection policy.
  - UCB1 (with slight modifications) balances exploitation and exploration remarkable well.
  - The Fear Of Missing Out is an *excellent* motivator for exploration.
- UCT is the combination of MCTS and UCB1, and is an *extremely successful* algorithm.
  - Yet it has obvious shortcomings,
  - There are alternatives to FOMO to motivate exploration, such as ϵ-greedy and softmax.