Monte Carlo Tree Search

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : easwar.subramanian@tcs.com / cs5500.2020@iith.ac.in

Novemer 04, 2022

Overview

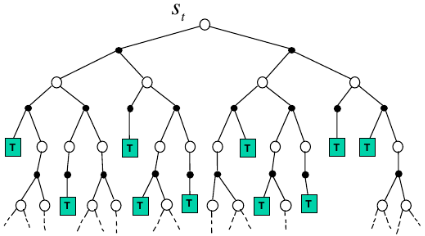
1. Introduction
2. On Truncated Tree Search
3. Naive Approach
4. Monte Carlo Tree Search
5. Derivative Free Methods

Introduction

Introduction

- We consider board games; Specifically, two player zero sum perfect information board games
  - ★ **Zero Sum :** Each participant's gain or loss is exactly balanced by the losses or gains of the other participant
  - ★ **Perfet Information :** No hidden information. During game-play every player can observe the whole game state.
- **Forward tree search** methods are popular to arrive at optimal moves in such board games
- Forward search algorithms select the best action by **lookahead**
- Lookahead is done using the model of the game MDP
- Apart from two player perfect games, tree search methods (such as MCTS) are used in situations where online planning using search is possible

Tree Search Methods : Framework



1. In most games, when described as MDP, there is no randomness in the environment; Moves are 'fullfilled'
2. Build a search tree with the current game position as the root
3. Compute value functions using simulated episodes
4. Select the next move to execute based on simulated epsiodes

Above framework is an example of online planning with search !!

Slide Credit: Katerina Fragkiadaki : CMU 10703

On the need for Online Learning

Question : Why can't value functions be learnt offline ?

- Environment has many states (Go : 10^{170}; Chess : 10^{48})
- Hard to compute a good value function for each one of them

Solution :

- Search tree is built with current game position and try to estimate the value function
- Solve the sub MDP ($\mathcal{M}^v$) starting from current game position
  - ★ Simulate episodes from current game position and apply model-free RL to simulated episodes

Slide Credit: Katerina Fragkiadaki : CMU 10703

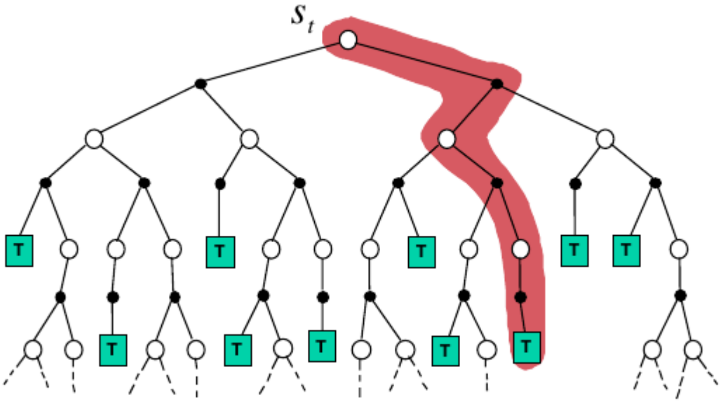
On Truncated Tree Search

Intelligent Vs Exhaustive Search

- The sub-MDP rooted at the current game position may still be very large
  - ★ More actions $\rightarrow$ Large Branching Factor
  - ★ More steps $\rightarrow$ Large Tree depth
- Reduce the breadth of the search by sampling actions from a policy $\pi(a|s)$ instead of trying every action
- Reduce depth of the search tree by position evaluation
  - ★ Truncate the search tree at state s and replacing the subtree below s by an approximate value function $V(s) = V^*(s)$ that predicts the outcome from state s

Slide Credit: Katerina Fragkiadaki : CMU 10703

Intelligent Vs Exhaustive Search



Contrast with Minimax and Alpha-Beta pruning !!

Figure Source: David Silver : UCL

Position Evaluation

- Engineer them using human experts (Example : DeepBlue !!)
  - Replication across domain not possible
- Learn from self play

Naive Approach

Position/Action Evaluation using Monte Carlo

- Simulate K episodes of experience from the current board position with the model

$$\{s_t^k, a_t^k, r_{t+1}^k, s_{t+1}^k, a_{t+1}^k, r_{t+2}^k, \cdots, s_T^k\}_{k=1}^K \sim \mathcal{M}^v$$

- Apply model-free RL to the simulated episodes

State Value Function Evaluation : Monte Carlo

Algorithm Evaluate Given Board Position using MC

Let K be the number of simulations
Let s be the current state ; Initialize $w = 0$ and $l = 0$
for $k = 1, \cdots, K$ **do**
    $s' \leftarrow s$
    while s' is non-terminal **do**
        Choose an action a (using possibly a random policy) that is admissible from state s';
        Take action a from state s' and store next state in s'
    end while
    if game won **then**
        w++
    else
        l++
    end if
end for
Return $(w - l)/(w + l)$

Action Value Function Evaluation : Monte Carlo

- Given a model $\mathcal{M}^v$, current board position s_t and **simulation policy** π
- For each action $a \in \mathcal{A}$
  - Simulate K episodes of experience from the current board position with the model

$$\{s_t^k, a_t^k, r_{t+1}^k, s_{t+1}^k, a_{t+1}^k, r_{t+2}^k, \cdots, s_T^k\}_{k=1}^K \sim \mathcal{M}^v, \pi$$

  - Calculate accumulate total reward and use it to compute action value estimate

$$Q(s_t, a_t) = \frac{1}{K}\sum_{k=1}^{K} G_t$$

$$\frac{1}{K}\sum_{k=1}^{K} G_t \xrightarrow{P} Q^\pi(s_t, a_t)$$

- Select action with maximum Q value

$$a_t = \arg\max_a Q(s_t, a)$$

Monte Carlo Tree Search

Improvements to Simulation Policy

Question :
With more simulations, how can we improve the simulation policy ?

Answer :

- We can keep track of action values (Q) not only for the root but also for nodes internal to a tree we are expanding!
- How should we select the actions inside the tree ?
  - ★ Use exploration algorithm(s) that we learnt in Bandit lectures
  - ★ Specifically, we could use the variant of the UCB1 formula given by,

$$a_t = \arg\max_a \left[\underbrace{Q(s_t, a)}_{\text{Exploitation}} + \underbrace{c \cdot \sqrt{\frac{\log N}{n_a}}}_{\text{Exploration}} \right]$$

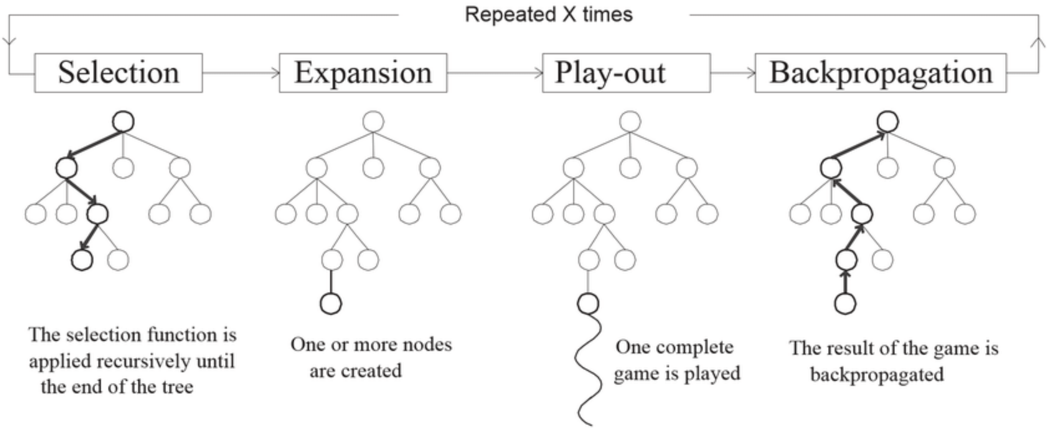
where N is the number of times the parent node is visited and n_a the number of times action a has been picked

Monte Carlo Tree Search

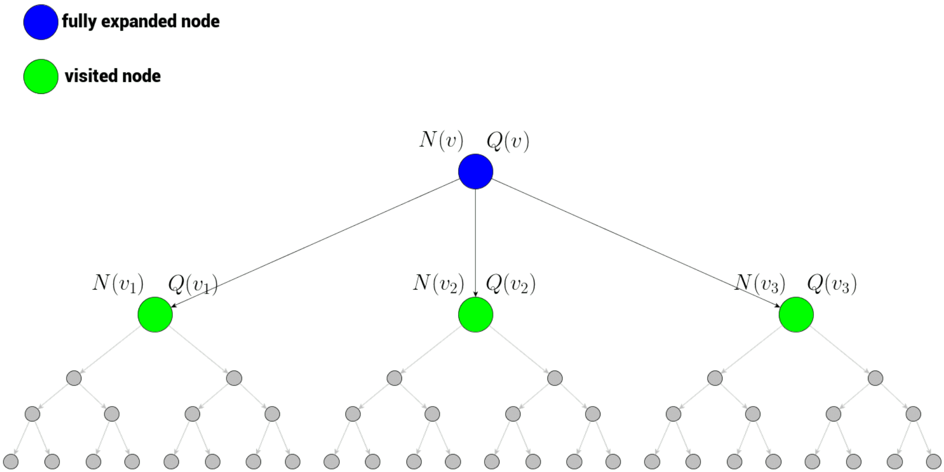
- Selection
  - ★ Used for nodes we have seen before
  - ★ Pick according to UCB
- Expansion
  - ★ Used when we reach the frontier
  - ★ Add one node per playout
- Simulation
  - ★ Used beyond the search frontier
  - ★ Don't bother with UCB, just play randomly
- Backpropagation
  - ★ After reaching a terminal node
  - ★ Update value and visits for states expanded in selection and expansion

Slide Credit: Katerina Fragkiadaki : CMU 10703

Monte Carlo Tree Search

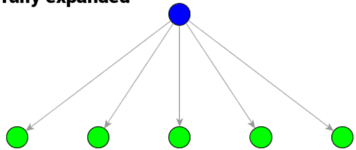


MCTS : Selection

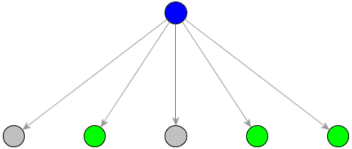


MCTS : Expansion

all children are marked visited - node is fully expanded

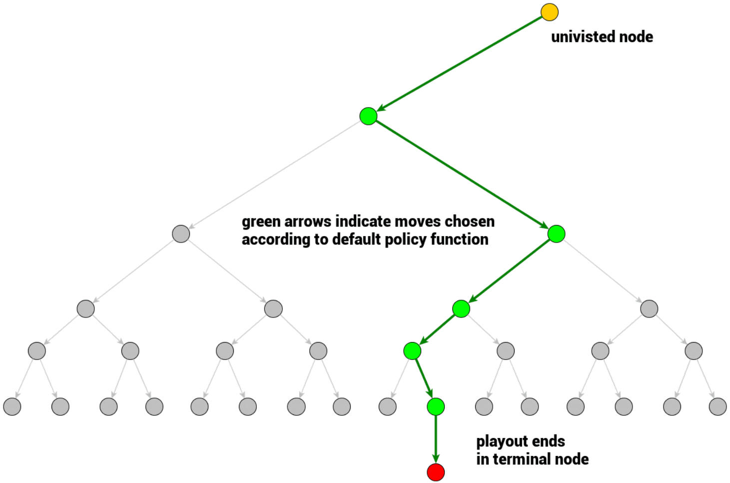


simulation/game state evaluation has been computed in all green nodes, they are marked visited

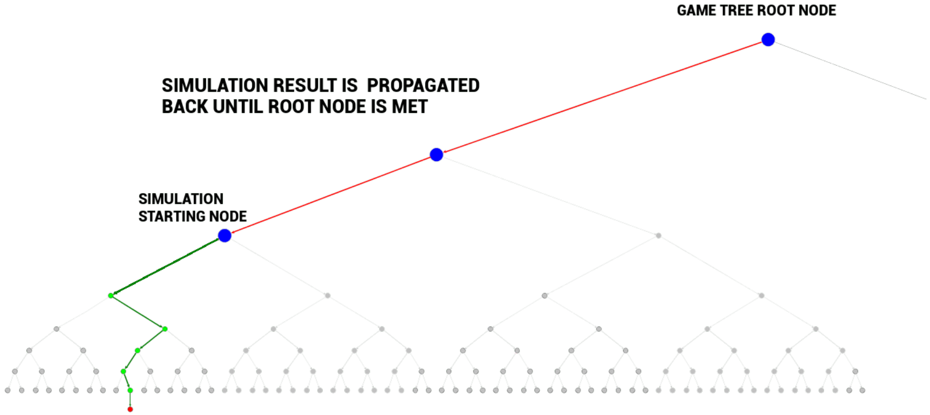


there are two nodes from where no single simulation has started - these nodes are unvisited, parent is not fully expanded

MCTS : Simulation



MCTS : BackPropagation



MCTS : Algorithm Sketch

Algorithm MCTS : Input 'node'

for $k = 1, \cdots, K$ **do**
    leaf = TRAVERSE(node)
    simresult = ROLLOUT(leaf)
    BACKPROPAGATE(leaf, simresult)
end for
Return 'best' child of 'node'

Algorithm TRAVERSE : Input 'node'

while node is fully expanded **do**
    node = SELECTION(node)
end while
if some children of node is not expanded **then**
    node = RANDOMUNEXPANDEDCHILD(node)
end if
Return node

MCTS : Algorithm Sketch

Algorithm SELECTION : Input 'node'

for all children of node **do**
    $\text{UCB[child]} = \text{child.value} + C \cdot \sqrt{\frac{\log(\text{node.VISITS})}{\text{CHILD.VISITS}}}$
end for
Return child with maximum UCB[child]

Algorithm ROLLOUT : Input 'node'

if node is TERMINAL **then**
    Return result
else
    child = PICKRANDOM(node.children)
    Return RANDOMPLAYOUT(child)
end if

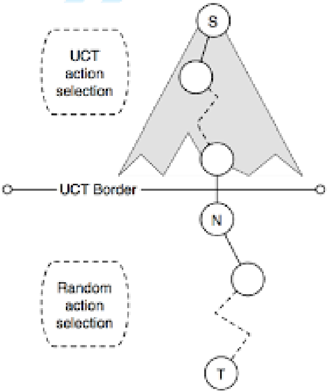
MCTS : Algorithm Sketch

Algorithm BACKPROPAGATE : Input 'node' and 'result'

```
if node is root then
    Return
else
    node.stats = result
    BACKPROPAGATE(node.parent)
end if
```

- The above pseudo-code is only a sketch. Please work out the details.
- For example, updating **'stats'** could involve incrementing number of visits to the node (needed for UCB computation) and augmenting the game results (win vs loss) from that node (needed to compute 'best' child)

Monte Carlo Tree Search



UCT (Upper confidence bound for Trees) based sampling of actions make the MCTS looks at more interesting moves more often

On Choice of Best Action

- How many simulations to run ?
  - ★ Time based : Run as long as you can
  - ★ Number based : Run K number of simulations
- When out of time, which move to play?
  - ★ Highest mean reward (highest probability to win)
  - ★ Highest UCB
  - ★ Most simulated move

AlphaGo : Successful Application of MCTS

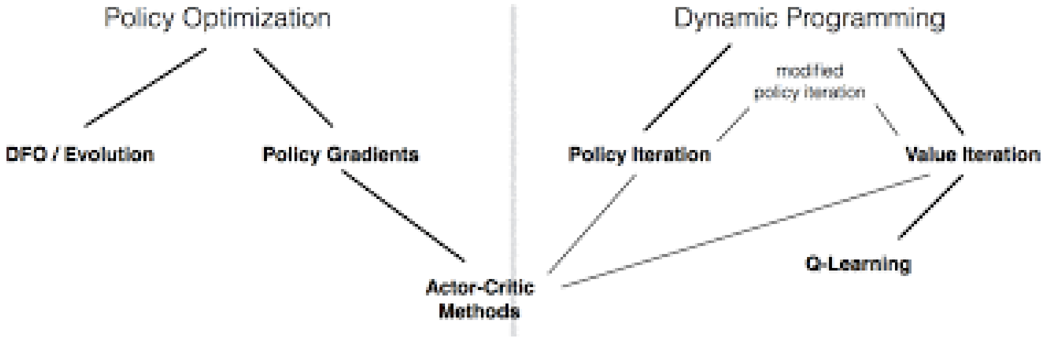
- Value neural net to evaluate board positions
- Policy network to suggest actions
- Combine those networks with MCTS

MCTS : Strength and Impact

- One of the advantages of MCTS is its applicability to a variety of games, as it is domain independent
- Basis for extremely successful programs for games and many other applications
- Very general algorithm for decision making
- Anytime algorithm $\rightarrow$ can be stopped anytime, although with time results improve

Derivative Free Methods

RL Landscape



Evolutionary Methods

Goal of RL is to find a policy π_θ^* such that

$$\pi_\theta^* = \arg\max_{\theta} J(\theta) = \arg\max_{\pi_\theta} \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right]$$

General Algorithm

- Start with an initial parameter θ and construct a policy and evaluate $J(\theta)$
- Make some random changes to the parameter and evaluate $J(\theta)$
- If the result improves, keep the change
- Else **repeat**

Cross Entropy Method

Algorithm Cross Entropy Method

Initialize policy network π with parameters θ_1
for $i = 1$ to N **do**
    Sample K parameters $\theta_{(i)}$ from a distribution $P_{\mu_i}(\theta)$
    Execute roll-outs for each of the K parameters
    Store $(\theta_i, J(\theta_i))$
    Select the top $p\%$ of the parameters θ in terms of the utility $J(\theta)$
    Fit a new distribution $P_{\mu_{i+1}}(\theta)$ from the top $p\%$
end for

- **Evolutionary** : The top $p\%$ of the parameter samples survive and the rest die. The top $p\%$ are then used arrive at the next generation of parameter samples
- **CMA-ES** : A popular variation that shrinks and expands the search area in the parameter space while fishing for parameters based on whether we are close to a good optima