Monte Carlo Tree Search

Andreas Orthey

March 2021

Monte Carlo tree search (MCTS) is a search algorithm for sequential decision problems, which tries to estimate the best available decision (action) to take. As a prototypical example, you could think about the game of chess, which you play by making a sequence of actions. MCTS is one of the most successful algorithms to solve such sequential decision problems.

To understand MCTS, you should be familiar with tree datastructures [Tree (data structure)(2018)] and with Upper Confidence Bounds (UCB). MCTS operates on a search tree, which is, in our case, a tree in which every vertex represents a state (e.g. a configuration of a chess board) and every edge represents an action (e.g. a chess piece move). Note that you can often not *explicitly* compute this search tree because there are too many states and it might take you years to create it. MCTS tackles this problem by only using stochastic sampling of paths through this tree.

To find the best available action, the basic idea of MCTS is to iterate over the search tree while a time budget is fulfilled. In one iteration, we choose an action (using a *tree policy*), expand the states for this action (*expansion* step), then follow random paths through the tree (using a *rollout* policy), estimate a reward (binary win/loss or a value) and use this reward to update the utility of the actions in the tree (the *backup* step). Let us write down the complete algorithm in pseudocode. As input, we use a tree V which can be empty or filled with nodes and values from previous runs.

Algorithm 1 MONTE CARLO TREE SEARCH

Input: Tree V, time budget
Output: Best child of root node

```
while time budget is fulfilled do
    leaf-node ← TreePolicy(root(V))          ▷ Choose a leaf node of V
    node ← Expansion(leaf-node)              ▷ Create a new child node (either random or using heuristic)
    Δ ← RolloutPolicy(node, V)               ▷ Rolls out a full simulation, with return Δ
    Backup(node, Δ)                          ▷ Update the values of node and all its parents
end while
return BestChild(root(V))
```

To run MCTS, you give it a time budget, i.e. for how long the algorithm should run to estimate the action utilities. The longer you run, the better your estimate is and the better your action becomes. Each iteration of MCTS depends on four methods, which are TreePolicy, Expansion, RolloutPolicy and Backup. Let us explain each method in more detail.

1 Tree Policy

In the tree policy method, we traverse the tree by starting at the root node and end up at a leaf node. A leaf node is a node at which there exists at least one child node from which we have not yet started a rollout (often called a simulation or a playout) [1]. To traverse the tree, we need to choose a child node v_c from every parent node v. For this decision, we use the upper confidence bound, i.e. the child node is chosen as

$$v_c = \underset{v' \in \partial v}{\operatorname{argmax}} \frac{Q(v')}{N(v')} + \beta \sqrt{\frac{2 \log N(v)}{N(v')}}, \tag{1}$$

whereby $N(v)$ are the total number of rollouts from that vertex in the tree, $Q(v)$ is the cumulative reward obtained over all rollouts, β is a parameter to trade-off exploitation and exploration and ∂v returns the child nodes of v.

[1] Note that some MCTS variants define a leaf node as a node which has no child nodes. An expansion step would then expand all child nodes simultaneously and start rollouts in parallel. We, however, do not do that, since it might be too costly if the number of actions is large.

2 Expansion

In the expansion step, we use the leaf node v to create a new child node. Usually, we just pick a random action a from the set of actions at v. You can, however, also use heuristics as a smart action selection strategy.

3 Rollout Policy

Starting at the newly expanded node, we use a *rollout policy* to traverse the tree until we reach a terminal vertex and receive a reward Δ. This reward can be binary (win/loss) or a value indicating your performance. To execute the rollout, you can either use a uniform rollout policy, where you play a random action. Or, you can use specialized heuristics to make informed estimates on the next best move. In practice, the performance of MCTS can differ dramatically based on which heuristic you use.

4 Backup

Once you receive a reward, you need to update the expansion node and all its parents. This is done in the backup step. Starting at our last expanded node v and the reward Δ from the rollout, we traverse the tree upwards towards the root node while adding the reward to each node. The pseudocode is shown in Tab. 5.

Algorithm 2 BACKUP

Input: Node v, Tree V, Reward Δ
Output: Update all parents and itself

while v is not equal to root(V) **do**
    $N(v) \leftarrow N(v) + 1$
    $Q(v) \leftarrow Q(v) + \Delta$ ▷ Store cumulative reward
    $v \leftarrow \text{PARENT}(v)$ ▷ Traverse Upwards
end while

If you want more information, I can also recommend the excellent Wikipedia article: [Monte Carlo tree search(2019)].

References

[Monte Carlo tree search(2019)] Monte Carlo tree search. Monte carlo tree search — Wikipedia, the free encyclopedia, 2019. URL `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`.

[Tree (data structure)(2018)] Tree (data structure). Tree (data structure) — Wikipedia, the free encyclopedia, 2018. URL `https://en.wikipedia.org/wiki/Tree_(data_structure)`.