Monte Carlo Tree Search for game play and portfolio selection

Explanation methodology

We first explain how Monte Carlo tree search works and then we project it on our problems. We use the notation and illustrations by Wikipedia.

The UCT1 ratio

The UCT is an acronym for upper confidence bounds applied to trees. It's used in node selection, the first step in Monte Carlo tree search as we show in the next section. We first explain it as it is from Wikipedia in terms of gameplay and then explain how we're going to use it to maximize the financial utility.

$$UCT1(node) = \frac{w_i}{n_i} + c\sqrt{\frac{ln(N_i)}{n_i}}$$

- w_i stands for the number of wins for the node considered after the *i*-th move
- n_i stands for the number of simulations for the node considered after the *i*-th move
- N_i stands for the total number of simulations after the *i*-th move run by the parent node of the one considered
- *c* is the exploration parameter. It's used for experimenting between exploration vs exploitation. A high c would give a rise to exploring what we didn't try before. A low c will be more focused on looking at w_i/n_i, i.e. what happened in the past.

Monte Carlo tree search algorithm

For each iteration:

- Selection: Choose a node that maximizes the UCT1
- Expansion: If the node we're at isn't a node from the final stage, i.e. it ends the game, create one or more nodes from it, i.e. branching operation.
- Simulation: Complete one random playout from the node that we created. Keep simulating the unknown factors until we reach our final stage when the game ends.
- Back Propagation: Using the results of many many simulations we should have a path that has yielded the most wins using the simulation. This is the path that we should choose at every stage when we're prompted to take an action.

An example

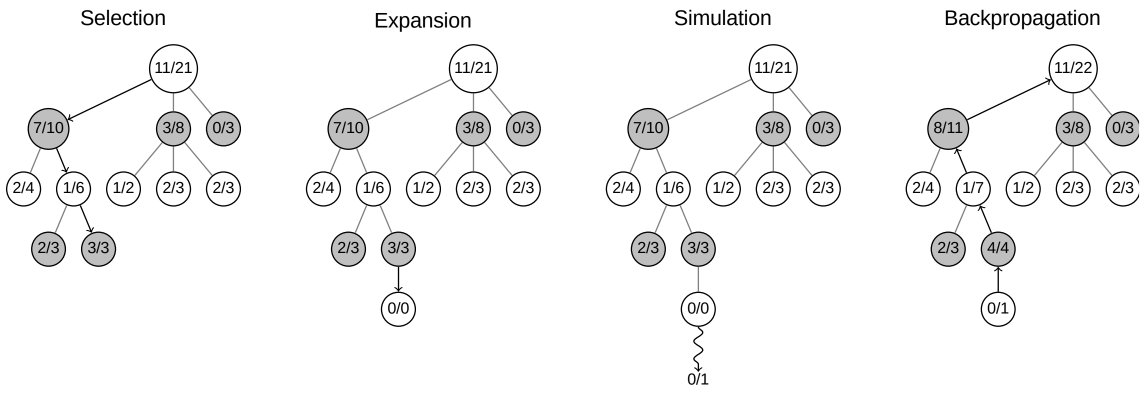
This example combines Minmax algorithm with Monte Carlo tree search so the wins in every turn are inverted (considering zero-sum games). The nodes for a certain turn (a certain player, assuming we have two players) are colored. Say light color belongs to player one, and dark color belongs to player 2.

For every node, there's a ratio. It's the ratio representing $\frac{w_i}{n_i}$ where w_i is the number of wins and n_i is the number of simulations that've happened through this node. So, every node's simulation is the sum of all simulations through its children.

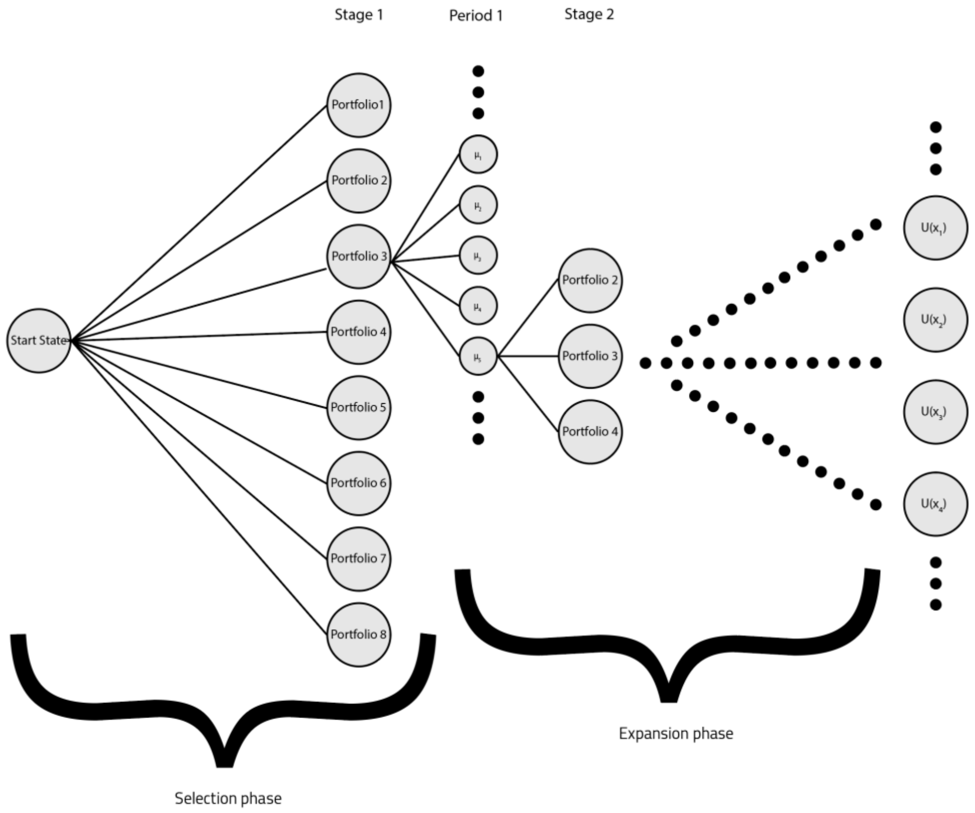
During node selection, we choose the node that maximizes such ratio

$$UCT1(node) = \frac{w_i}{n_i} + c\sqrt{\frac{ln(N_i)}{n_i}}$$

In the following example it becomes UCT1(node 3 / 3) = 3 / 3 + c * sqrt(ln(6) / 3).



Now, maximizing the financial utility by selecting different portfolios at each stage.



An illustration of Monte Carlo tree search after a few iterations.

Notes

- Note that in our application, since we don't use Monte Carlo tree search for games, we replace w_i with the utility U(x) where x is the amount of money.
- We're also not playing against another player, so all aspects of Minmax are eliminated.

- We don't know what's exactly going to happen in the future. However, we do have a pdf of each portfolio's returns: N(μ, σ) that can easily be calculated from the given data about returns per sector and their correlation.

Algorithm

1. Selection: We start at the start state and we select one of the portfolios. Say we select portfolio 3 based on UCT1.
2. Expansion: We sample from the distribution of the returns: N(μ, σ) and create a node for a "scenario" as we see in the illustration for μ_5.
3. Simulation: We then construct the following stage by expanding all our possibilities, i.e. say portfolio 3, we playout this until we reach a terminal state where we have a value for the utility function. We update the nodes going back, i.e. Portfolio 3 node and μ_5 node.
4. We keep doing steps 1, 2, 3 for many many iterations until the value function doesn't change (epsilon comparison).
5. Back Propagation: Based on our expected utility value (wins in the original algorithm), we can learn what action to choose at every stage.

Applying these steps for one stage

- Selection: Using $UCT1(node) = \frac{w_i}{n_i} + c\sqrt{\frac{ln(N_i)}{n_i}}$, all nodes return ∞, since n_i is initially zero, so we choose any node (portfolio) randomly. $n_i = 0$ means that no simulations have been done in this node's children.
- Expansion: This is a technicality of the algorithm's implementation, but we don't expand the first time we simulate through a node. The logic for this case is given in the appendix.
- Simulation (or rollout): We, then, keep simulating a stage after another (and periods too) until we reach a terminal state. In stages, we choose portfolios. In periods, we simulate returns. We do this until we reach the final stage and have a utility. We update the node we choose with n_i = 1 and w = U(x)
- Back Propagation: [Not applicable for one iteration] After many simulation k, enough to expand all our nodes from stage 1 to the final stage, we're able to back propagate in such a way that maximizes the expected utility.

Few comments

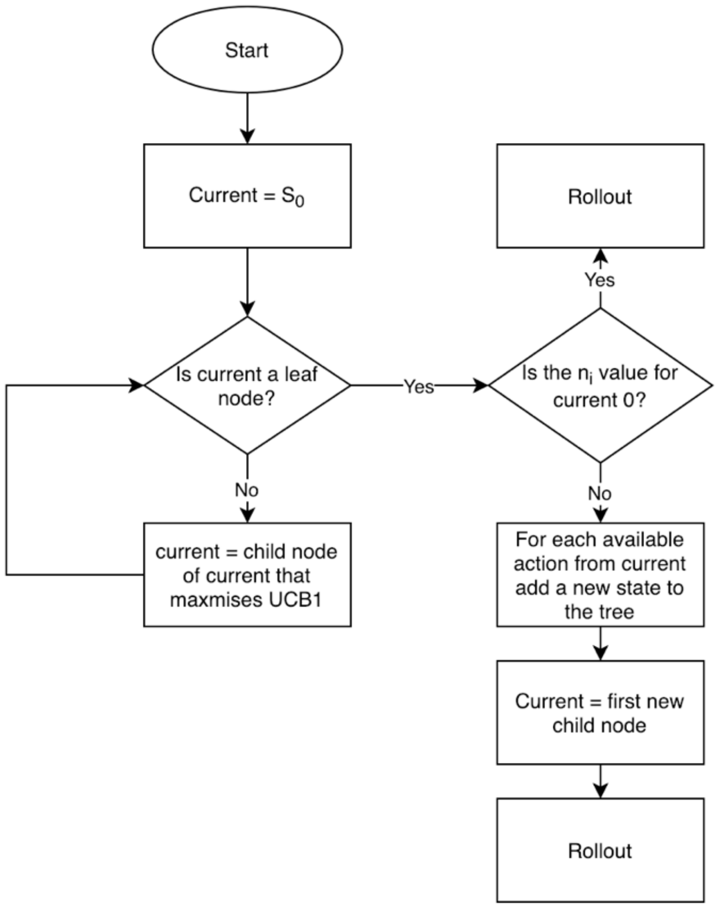
- Usually researchers, re-assess the state of the world when it's time to make a new decision. So, this algorithm will be run once a year in our case.
- The value function is sometimes approximated using a neural network if no domain knowledge can be applied.

References

- Wikipedia

Appendix

- When to expand then simulate and when to just simulate?


Start
Current = S_0
Is current a leaf node?
No
current = child node of current that maxmises UCB1
Yes
Is the n_i value for current 0?
Yes
Rollout
No
For each available action from current add a new state to the tree
Current = first new child node
Rollout