How to apply Monte Carlo Tree Search to design a search algorithm?

There are three steps in which we are going to answer this question. First, what does our algorithm search for? Second, what framework will we design our search algorithm in? And third, what exactly does the tree look like to which we apply Monte Carlo Tree Search?

What are we searching for?

Suppose, we have a string $x \in \{0,1\}^n$. We define its quality to be the length of the longest prefix of ones, for example quality$(111010) = 3$ and quality$(011111) = 0$. Formally,

$$\text{quality}(x) = \sum_{l=1}^{n} \left(\prod_{i=1}^{l} x_i \right).$$

Apparently, the optimal string is the all-ones string.

What are the choices in designing our algorithm?

Our algorithm will see only the quality of the string x. That is our first restriction on the design space. Second, we simplify the algorithm design by allowing the algorithm only to sample the next string y uniformly at random at a 1-norm distance of r from the previous string x. That is, we sample around x, at a chosen radius r. These radii can be restricted to a set $\{r_1, \ldots, r_k\}$.

So, the algorithm starts with a string x drawn uniformly at random from $\{0,1\}^n$. The algorithm checks its quality. Based on this quality q, the algorithm triggers a certain radius $\pi(q) = r$ to be used for the sampling of the next string y. π is the fixed policy of choosing a radius. The algorithm terminates, once a string with quality n has been found. Our goal is to minimize the number of sampled strings until that happens.

The policy π is the design space of the algorithm. This is what we want to optimize.

What does the tree for MCTS look like?

Let's define a game from all of the above: We will define the observations, the actions, the transitions, the rewards.

The *observations* are the qualities, 1 through n. The states behind these observations are strings in $\{0,1\}^n$.

The *actions* are the sample radii, r_1 through r_k. For example, we might have only two radii $r_1 = 1$ and $r_2 = 5$.

The *transitions* consist in flipping the bit in exactly r positions in x to reach y. The bit positions are chosen uniformly at random.

The *rewards* are always -1 for every transition taken. So, we want to minimize the number of transitions until we find the optimal string.

This is it. Now, our tree is built by first putting the current string x in the root node. Then we apply a Monte Carlo Tree Search algorithm and build out a tree. Eventually, we use the collected statistics to choose a radius and sample the next string.

Extensions

The design space can be extended later by allowing the algorithm to see the entire string x instead of only its fitness. Furthermore, more complex algorithms can be regarded which keep in memory a population of strings rather than only one string and which have more sampling parameters rather than only a radius.

Getting started

What do you think are important properties of this design game and which specific MCTS algorithm would you try first?