1 Monte Carlo Tree Search algorithm

Monte Carlo Tree Search (MCTS) algorithm is a family of algorithm based on Monte-Carlo simulations and a tree seach procedure. Monte-Carlo simulations were first used in 1993 [2] for the game of Go. In 2006 [4], MCTS most popular algorithm called Upper Confidence bound applied to Trees (UCT) was introduced. It used multi armed bandit in the tree search procedure to keep a balance between exploitation and exploration of the tree. Which, respectively means to study more a well known and good strategy and explore new stategies. Nowadays, MCTS algorithms have been used in many domains [1] and still give state of the art result in many of them. For example, it is a core component of algorithms such as AlphaZero [6].

Here, we are interested in solving optimization problems, that can also be seen as games or puzzle. Moreover, we have a particular interest in stochastic problems. In this section, we will discuss the different classic components of a MCTS algorithm and we will present UCT and its evolution Generalized Rapid Action Value Estimation (GRAVE). We will first discuss the preliminaries to understand the MCTS.

1.1 Markov Decision Process

Optimization problem solved using MCTS algorithm are modelised as a Markov Decision Process (MDP). it modelises the problem as a set of states and actions that can be applies or played on a state. A state contains all the information necessary to understand the situation meaning that the past state are irrelevant. In this section, we will give a vulgarize explanation of what a game is and a more definition of a MDP.

1.1.1 Sequential problem modelization

Let's consider a single player basic game called the Left Most Problem (LMP). Its rules are simple:

- Each turn the player can choose *left* or *right*.
- A player choosing *left* gain one point.
- A player choosing *right* gain no point.
- The game ends after N turn.

The goal of the game is to have as much point as possible at the end. It is a very simple problem but is perfect to explaine the key concept. A state of the game can be represented by two elements: the score of the player and the number of turn that passed. So if we define the state $s = (6, 10)$ it means that the player has 6 points and has played 10 turns. Past states are irrelevant to describe s, it contains all information necessary to describe the situation. Some states are called terminal because they correspond to state where the game has ended. Here, all the states where the player has played N turns are final or terminal.

The actions the player can choose are *left* and *right*. If the player applies *left* to s then, we will obtain a new state $s' = (7, 11)$. The transition from s to s' is always the same if we apply *left* to s.

The goal being to obtain the most point as possible, a player will learn to always pick *left*. This is called a policy or, more conviniently, a strategy: it dictates the action according to a situation. In other, if I give s to a player following the policy to always go right, he will choose *right* and will go to state $s'' = (6, 11)$.

This simply explains the key concepts of a MDP. You have states on which you apply actions. Applying certain actions on certain states will give you a new state. And states have what is called a reward, which is the number of point the player scored.

1.1.2 Definition

Now for a more formal definition, a MDP is composed of:

- S a set of state.
- A a set of actions
  - $A_s \subseteq A$ are the legal actions or actions that can be applied to s
  - If $A_s = \emptyset$ then it means that s is a final or terminal state
- $P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that when applying action a to state s, we obtain state s'
- $R_a(s, s')$ is the reward obtained when transitioning from s to s' by applying action a

When optimizing a MDP, we search for the best policy, that means the policy that will give the best expected reward possible. A policy $\pi(s)$ is a distribution of probability over all the elements of A_s. A policy that maximizes the reward is called an optimal policy and is often noted π^*.

A MDP represents a stochastic game if the probabilities given by $P_a(s, s')$ are not equal to 1. That is to say, if the transitions between the states are uncertain, therefore random, the modelized problem is stochastic.

1.2 Key steps of a MCTS

A MCTS algorithm such as UCT follows four steps that are repeated indefinitely. Theses steps are the selection, the expansion, the simulation and the backpropagation. Here, we will give a quick explanation of each of these steps. But, before going in, we have to explain what the tree used in MCTS approach looks like.

A MCTS tree is a tree as defined in graph theory which is not mandatory to explain here. At the top of the tree, we find the root state, which is the current state of the game, the one for which we search the best action. Each node of the tree is a state and nodes are connected by actions.

During the selection the algorithm will descend in the tree from node to node, starting from the root node. At one point, according to a policy, it will stop on a node corresponding to a state s and begin the expansion phase.

The expansion phase corresponds to the addition of a new node on the tree. It applies an action $a \in A_s$ chosen according to a policy. Then, it adds the resulting state, let's say s' to the tree.

The simulation is a Monte-Carlo simulation. It is used to evaluate the quality of a node for a given player. It consists in playing the game randomly from state s' until a terminal state is reached. No modification is done to the tree. The simulation is called a rollout because it is a random playout. It follows what is called the default policy which chooses the action uniformly.

The backpropagation is the last phase. The result of the rollout is transmitted back to the top of the tree. It follows back the way chosen at the selection. We can then have statistics on each node which, often, are useful to create the policy used during the selection and expansion.

All theses steps are repeated over and over again. The number of repetition is often limited by a search time which is set beforehand. Based on the tree obtained at the end, the player can choose which action apply on the root state. Often, the search is repeated on a new tree each turn to avoid the algorithm to be stuck on the same strategy. erge

uct and
grave or
remove al
saying ab
them

2 The Nested Rollout Policy Adaptation

The Nested Rollout Policy Adaptation (NRPA)[5] is a MCTS algorithm that has been used to solve optimization problem with state of the art results like the Traveling Salesman Problem with Time Window (TSPTW)[3]. Howether, it is not a traditional MCTS algorithm as it does not follow the classic steps of these algorithms: selection, expansion, simulation and backpropagation (see section 1.2). Even the tree procedure is very different from other MCTS algorithm. The NRPA learns a playout policy used during the rollouts. Actions are associated to weights and sampled during rollout according to the exponential of this same weight. Doing so, it find the best sequence of actions it can and modifiy its policy to adapt it to the best known sequence. In the end, it returns the best sequence found.

This algorithm is very effective to solve non stochastic optimization problem and has found new records for game like Morpion Solitaire. In this section, we will explain the NRPA in three parts: the playout, the adaptation and the core function.

2.1 NRPA playout

A NRPA playout is used to generate and evaluate a sequence of actions. So a sequence $S = a1, a2, a3, \dots$ is a list of actions that lead from the root state to a terminal one.

References

[1] C. B. Browne et al. “A Survey of Monte Carlo Tree Search Methods”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43. DOI: `10.1109/TCIAIG.2012.2186810`.

[2] Bernd Brügmann. “Monte carlo go”. In: (Nov. 1993).

[3] Tristan Cazenave and Fabien Teytaud. “Application of the Nested Rollout Policy Adaptation Algorithm to the Traveling Salesman Problem with Time Windows”. In: *Learning and Intelligent Optimization*. Ed. by Youssef Hamadi and Marc Schoenauer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 42–54. ISBN: 978-3-642-34413-8.

[4] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *Proceedings of the 17th European Conference on Machine Learning*. ECML’06. Berlin, Germany: Springer-Verlag, 2006, pp. 282–293. ISBN: 354045375X. DOI: `10.1007/11871842_29`. URL: `https://doi.org/10.1007/11871842_29`.

[5] Christopher Rosin. “Nested Rollout Policy Adaptation for Monte Carlo Tree Search.” In: Jan. 2011, pp. 649–654. DOI: `10.5591/978-1-57735-516-8/IJCAI11-115`.

[6] David Silver et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. In: *CoRR* abs/1712.01815 (2017). arXiv: `1712.01815`. URL: `http://arxiv.org/abs/1712.01815`.