Monte Carlo Tree Search and Its Applications

Max Magnuson
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
magnu401@morris.umn.edu

ABSTRACT

Keywords

Monte Carlo Tree Search, Heuristics, Upper Confidence Bounds, Artificial Intelligence

1. INTRODUCTION

In 1997 the field of artificial intelligence(AI) experienced a monumental breakthrough when IBM's Deep Blue defeated Garry Kasparov, a reigning grand master, in a chess match[**?**]. They were able to achieve this by using brute force deterministic tree searching methods combined with human knowledge. The human knowledge allows for the computer to evaluate moves properly, and then populate a tree to search for the best move. This event really demonstrated to the world the power of computers and artificial intelligence.

While computers are capable of outplaying the top players of chess, they struggle when it comes to board games like Go[**?**]. Go is a board game that originated in China, and it has a strong professional community. It is a game about positional board advantage which is something traditional AI approaches struggle with evaluating. This is because moves in Go tend to have very long dependencies. A single move may have major effects on moves 50 to 100 moves down the line. Also Go has significantly more moves available to the player at anyone time than chess. These problems cause deterministic approaches to perform poorly. It is just too much for those approaches to efficiently handle.

People have started turning to alternative methods to approach Go. One such method, Monte Carlo tree search(MCTS) has had a lot of success in Go. MCTS eschews the typical brute force tree searching methods, and it utilizes statistical processes and heuristic approaches to decide what move to make. In 2009, for the first time ever, a computer defeated a top professional Go player in a 9x9 game[**?**]. It took twelve years since Deep Blue defeated Garry Kasparov for AI to achieve its first major victory in Go, and it was only on the smallest board that Go is played on.

MCTS has been growing in popularity in recent years, and it demonstrates a lot of promise. In this paper we will be examining MCTS and a few of its applications.

2. BACKGROUND

MCTS combines the random sampling of traditional Monte Carlo methods with tree searching. The random sampling is used to construct a game tree. This tree will be traversed based on statistical processes, and the MCTS method relies on the convergence of the tree to reliably choose the best move. MCTS is a heuristic method and as such it will not always find the most optimal move, but it has a reasonably high success of choosing moves that will lead to greater chances of winning.

2.1 The Tree Structure

MCTS structures the game state and its potential moves in a tree. Each node in the tree represents the state of the game with the root node representing the current state. Each line represents a legal move that can be made from one game state to another. In other words, it represents the transformation from the parent node to the child node. Any node may have as many children as there are legal moves. For example, at the start of a game of Tic-Tac-Toe the root node may have up to nine children. One for each possible move. Each following child can only have one less child than its parent since the previous moves are no longer available as options.

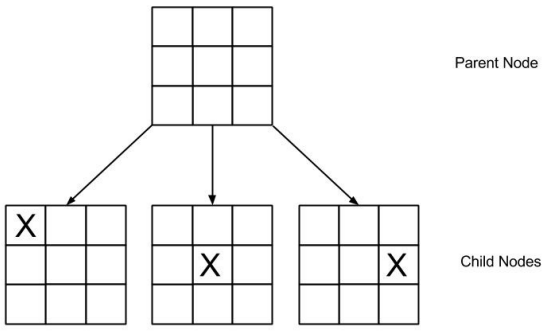

Figure 1: A small portion of what a tree represents

Figure 1 represents the top portion of a tree for the game Tic-Tac-Toe. The AI is making the first move, so the source

UMM CSci Senior Seminar Conference, May 2015 Morris, MN.

node is the first game board. Each child node represents the potential moves that can be made from the current game state. It is important to note here that those are only three of the potential nine child nodes.Todo: This somewhat contradicts what I say later on in the paper about UCT MCTS does not have to look at every potential child node, nor will it want to for sake of efficiency. Once MCTS has decided which move to make, the source node of the tree will then become the child it chose. For example, if MCTS chose the left child in figure 1, then the new MCTS tree would start at that child and everything else that was branching off of the original parent node is discarded.

Along with the game state, each node encodes for a value that represents how favorable choosing a particular node is. This value comes from the respective values of the nodes that branch off of it. In the example of Tic-Tac-Toe we could assign any simulation at a node that ends in a loss a zero, and any simulation that ends in a win a one. When any of these values are discovered, then the rest of the tree that branches into that node can be updated with its value. So in this case, choosing the node with the greatest value, leads to a path with the greatest ratio of wins to losses. By doing this, it gives the AI the greatest chance of choosing a winning outcome. This is what the MCTS algorithm relies on to be effective.

2.2 The Four Steps of MCTS

The process of MCTS is split up into four processes: Selection, Expansion, Simulation, and Backpropagation. These four processes are iteratively applied until a decision from the AI must be made.

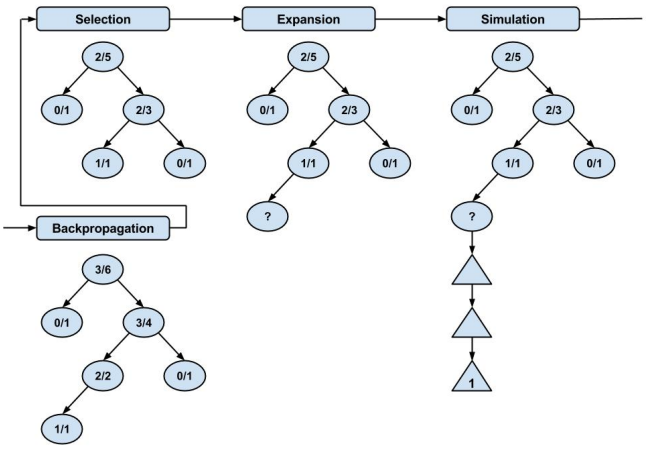

Figure 2: The four steps of MCTS

Todo: This section still needs a lot of work

Selection - In the selection process, the MCTS algorithm traverses the current tree using a tree policy. A tree policy uses an evaluation function that prioritize nodes with the highest estimated value. In 2 the MCTS algorithm traverses to the 2/3 node then the 1/1 because those are the nodes with the greatest estimated value.

Expansion - In expansion a new node is added to the tree as a child of the node reached in the previous step. There is only one node added to the tree in each iteration, and it is at this step. In 2 the newly added node is indicated by the ?.

Simulation - In this step, a simulation(also referred to as a playout or rollout) is played out according to the simulation policy[?]. The simulation may use either a weak or strong policy. A weak policy would use little to no predetermined strategy. It would simply playout the simulation randomly. A strong policy would use a more guided approach to choose the moves. A strong policy may make the simulation too deterministic or make it more prone for error[?]. The policy plays out moves until either an end state or a predefined threshold is reached. Then based on the result of the simulation, the value of the newly added node is established. For example, a simulation of a node for Go would reach the end of a game(the end state), and then determine a value based on whether the player won, or lost. In 2 the simulation ended in a 1. Therefore, the value of the new node is 1/1.

Backpropagation - Once the value of the node is determined, then as the name of this process implies, the rest of the tree can be updated. The algorithm will traverse back to the root node only updating the values of the nodes that it passes through. Only those nodes are effected because each node's respective value is an estimation of values of the nodes after them. In 2 there are two nodes that are left untouched since those do not branch into the newly added node.

2.3 Upper Confidence Bound Applied to Trees(UCT)

Todo: Get across that each unexplored node is explored before moving on, but only at each node that is visited. If a node is not visited again, that node's children will not be explored. The UCT is what the MCTS algorithm uses as a tree policy to traverse the tree. The goal of the UCT is to balance the idea of exploration versus exploitation. The concept of exploration promotes exploring many unexplored areas. This approach may explore many different paths on its way to finding the best decision. While this approach is useful to ensure that MCTS is not overlooking any potential paths, it can become very inefficient very quickly with games with a large number of moves. This is balanced with exploitation. The exploitation approach will tend to stick to one path that has the greatest estimated value. UCT balances these two ideas by exploring every unexplored node that it visits, but then it only visits nodes that yield the greatest estimated value.

$$UCT(node) = \frac{W(node)}{N(node)} + \sqrt[c]{\frac{ln(N(parentNode))}{N(node)}} \quad (1)$$

When traversing the tree, Equation 1 is applied to each of a node's children to evaluate the estimated value of that node[?]. N() represents the total number of simulations made at that node and the nodes branching off of it. W() represents how many of those simulations ended in a winning state. C represents an exploration constant that is found experimentally. The first part of the UCT takes into consideration the known estimated value of the node by determining the ratio of simulations won to total simulations. The second part of the UCT takes into consideration the unexplored nodes of the parent node by taking the ratio of the parent node's total simulations to the node's simulations.

3. USING MCTS TO PLAY GO