Artificial Intelligence, Assignment 1 Report

Zubair Abid, 20171076
Abhigyan Ghosh, 20171089
Team 24, X-Men

Monte Carlo Tree Search (MCTS) with Upper Confidence Bound (UCB)

1 Algorithm and Implementation

The Monte Carlo Tree Search is a general game playing search algorithm that performs reasonably well in turn based adversarial games without encoding any explicit domain based knowledge, by combining the best of randomness with the advantages of tree based search algorithms. It outperforms general minimax algorithms and their variants uniformly across the board, and can be extended to include several learning methods along with it. In this assignment, we shall implement just a basic version, and if time permits, use a hybrid of the MCTS with Minimax (Chen, Doan, Xu (2018)).

MCTS works by selecting random nodes in the search tree and assigning probabilities to states by playing out the game and backpropagating the result so that the optimal path can be taken when playing again. However, since a lot of the search paths are suboptimal and the time is limited, we shall be limiting the search tree using an Upper Confidence Bound for Trees (UCT) score (Kocsis and Szepervari (2006)). Together, these two algorithms combine to form our bot.

2 Advantages

MCTS with UCT offers a number of advantages over traditional Minimax, and other tree based variants.
The reasons for choosing this over that are elaborated in the Appendix.

1. *Does not require a heuristic*
   MCTS does not require any strategic or tactical knowledge about the given domain to make reasonable decisions. The algorithm can function effectively with no knowledge of a game apart from its legal moves and end conditions.

2. *Asymmetric focus on the search tree*
   MCTS performs asymmetric tree growth that adapts to the topology of the search space. The algorithm visits more interesting nodes more often, and focusses its search time in more relevant parts of the tree. This makes MCTS suitable for games with large branching factors such as 19x19 Go. Such large combinatorial spaces typically cause problems for standard depth- or breadth-based search methods, but the adaptive nature of MCTS means that it will (eventually) find those moves that appear optimal and focus its search effort there.

3. *Haltable*

The algorithm can be halted at any time to return the current best estimate. The search tree built thus far may be discarded or preserved for future reuse.

4. *Simple algorithm*
   The algorithm is simple to implement

3 Disadvantages

The algorithm comes with its own set of issues, too. Most of them are centered around the number of plays needed to maximise the utility of the randomness property of the algorithm.

1. *Potentially poor against Intelligent Agents*
   Chen, Doan, Xu (2018) showed that while the MCST has the best performance against randomised opponents, it could potentially loose out to a well trained intelligent agent. A possible solution to this is to setup the MCST by playing it against other intelligent agents along with against random agents.

2. *Playing Strength*
   The MCTS algorithm, in its basic form, can fail to find reasonable moves for even games of medium complexity within a reasonable amount of time. This is mostly due to the sheer size of the combinatorial move space and the fact that key nodes may not be visited enough times to give reliable estimates.

3. *Speed*
   MCTS search can take many iterations to converge to a good solution, which can be an issue for more general applications that are difficult to optimise. For example, the best Go implementations can require millions of playouts in conjunction with domain specific optimisations and enhancements to make expert moves, whereas the best GGP implementations may only make tens of (domain independent) playouts per second for more complex games. For reasonable move times, such GGPs may barely have time to visit each legal move and it is unlikely that significant search will occur.

4 Appendix

4.1 Motivation for Optimised Search

Xtreme Tic Tac Toe is a turn based, adversarial, perfect information game, so it is theoretically possible to construct a search tree for the entire game and search through it for the optimal solution. This can be achieved through an algorithm like Minimax, alternating between minimizing and maximising the utility as per the conflicting goals of alternating players. However, given the massive search space $(162 + 18^{97})$, WITH AN AVERAGE OF 98 MOVES PER GAME FOR RANDOM AGENTS, it is not feasible to traverse the entire search tree. There are two options here:

1. Pruning the search tree (with Alpha Beta pruning or something likewise), limiting the number of lookahead moves, and using a heuristic evaluation function to estimate a best choice.

2. Using a randomised probability based tree search algorithm: a decent method in case there exists no viable or hard to find viable heuristic for optimal performance

4.2 Monte Carlo over Minimax, the how and why

There aren't many optimal heuristics available for Xtreme/Ultimate Tic Tac Toe.
In this we consider 3 heuristics for a Minimax solution and then exhibit our preference for Monte Carlo:

1. This heuristic employs 4 overall strategies:
   "Favouring" one factor over another implies that the favoured factor yields a higher utility value.

   (S1): Favour a higher total count of the number of wins across all subgames.
   (S2): Favour specific positions within both the outer game board and inner boards. Specifically favour the middle position in any board over sides or corners. A distribution of "weighted marks" is assigned to each position to indicates favoured cells.
   (S3): Aim to complete a row, column, or diagonal.
   (S4): Ensure a zero-sum game by multiplying the utility value by a factor of -1 when computing the opponent's utility. This also enables proper use of alpha-benta pruning.

2. It is the same as Heuristic 1, except that (S3) is removed in favour of higher performance and a greater possible depth limit.

3. This evaluation function simply takes the number of "mini-boards" that the agent has won minus the number of "mini-boards" that the agent has lost.

Heuristics 1 and 2 are taken from an existing attempt at comparing multiple heuristics and their solutions: https://github.com/noahnu/ai-ultimate-tic-tac-toe/blob/master/README.md
The report and code provided demonstrably show the performance improvements with just a simple monte carlo algorithm over both the heuristics with a simple minimax.
Heuristic 3 is taken from another comparative report (Chen, Doan, Xu (2018)) and performs almost as well as the monte carlo, but is barely outperformed.