Frank Wu

260580792

Comp 424 Artificial intelligence project report

Approach Description:

The approach I chose for this project is Monte Carlo Tree Search. In order to implement the algorithm in a well-formed and simply manageable style, I create two classes in MyTool package named “Node” and “MonteCarlo”. I also create two helper methods called “UCB” and “RandomRollOut ” in myTools class.

The MonteCarlo class is the class do the primary implementation and calculation of the algorithm, it will build a tree structure gradually with Node class and helper method to return the optimal move within the user desired maximum running time.

The process of my program works like this: First I create an instance of MonteCarlo class in chooseMove method of StudentPlayer class with current board state , player id , the time when the instance is created and desired max running time for the algorithm as input of constructor, then just simply run the run( ) method under MonteCarlo class to get a new move as return to do the next move.

The Node class represents the basic data structure which is used to build a tree in MonteCarlo class. Each node contains a board state, a list of children node , a parent node, a move which is the move that its parent took to get to this node , number of times for this node or its children winning games in random roll out and number of visit which is the number of times for this nodes or its children doing random roll out. And the node class also contains getter function or IsLeaf function which are fundamental helper function for tree structure .

The main algorithm will take the input board state to build the root of the tree, then it will doing the process: selection -> expansion -> random roll out (simulation) -> back propagation and repeat this process until the maximum running time limit is reached. (which is calculated using the time when the instance of algorithm class is created) In the selection process, the function will traverse the tree based on selection policy until it reach the leaf of

the tree. The selection policy I used is Upper Confidence Tree. To select node with best value calculated based on the UCT formula. After reached the leaf of tree, the function will check wether the node has been visited or not by checking the node's number of visits. If yes, then add all possible legal moves as new nodes under the node and set those new nodes as children of the node. Then do the random simulation (or random rollout) on the first child node. Since the number of visit of new nodes are always 0 and therefore the UCT value for new nodes are infinite, the algorithm will always prefer to visit unvisited nodes. After getting result of random rollout, back propagate the result by adding the number of wins and visits to its parent and the parent of its parent recursively until reach the root. If the currently select node after selection has not been visit then do a random rollout and back propagate the result same as above. The random rollout is to let two players at the given state always pick random move from legal next move and check which player is the winner. Random rollout return 1 if the our player wins and 0 if opponent player wins. After expanding the tree and fill each node with updating wins and visits, we got a complete tree structure. Then the function will go back to the root and return the move of the node with greater value of wins/visits from children of root.

The motivation to choose this approach is the event when Google 's AI AlphaGo beat world champion of GO Lee Sedol in 2016 and AlphaGo also beat several professional top GO player later. As the first AI beat professional GO player in history, AlphaGo used the Monte Carlo tree search as one of its primary approach. I have learned GO and I know it is very complicated, so I was amazed by AlphaGo and its algorithm. I really want to understand and try the approach that AlphaGo used and thats why I pick this approach for this project.

Theoretical basis of approach:

From WIkipedia :

"The focus of Monte Carlo tree search is on the analysis of the most promising moves, expanding the search tree based on random sampling of the search space. The application of Monte Carlo tree search in games is based on many *playouts*. In each playout, the game is played out to the very end by selecting moves at random. The final game result of each playout is then used to weight the nodes in the game tree so that better nodes are more likely to be chosen in future playouts." [1]

So the algorithm is working based on the random playouts. This means Monte Carlo Tree Search form its own value function to judge if a certain move is good or bad by just simulating a game instead of analyzing how the move change the game state. The simulation by random would not give very accurate information about the move by its own, but the information will be promising when there are enough number of simulation come together after running for a decent amount of iteration.

The Upper Confidence Tree formula also make sure the algorithm will pick a promising move in the selection process.

Quote :

“Then, your strategy is to pick the machine with the highest upper bound each time. As you do so, the observed mean value for that machine will shift and its confidence interval will become narrower, but all of the other machines' intervals will widen. Eventually, one of the other machines will have an upper bound that exceeds that of your current one, and you will switch to that one. This strategy has the property that your *regret*, the difference between what you would have won by playing solely on the actual best slot machine and your expected winnings under the strategy that you do use, grows only as O(nln n)” [2]

Advantages and disadvantages of approach:

The biggest advantage of Monte Carlo tree search is that this method does not need an explicit evaluation function since it will just do random move simulation to and use the consequence of that game to determine which move is better. Since in practice, AI game search usually cannot get a very accurate evaluation function . Another advantage of the approach is that it is not affected by branding factor of the game.

Quote from course slides: [3]

“ -Unaffected by the branching factor (i.e. #actions):

-Control the number of lines of play, so move can always be made in time. If branching factor is huge, search can go much deeper, which is big gain.”

The disadvantage of the approach is that it may not be optimal and does not perform better than MiniMax on games with small branching factor.

Alternative approach:

While implementing Monte Carlo Tree Search, I also tried MiniMax search. MiniMax search is to first expand a complete search tree, then start from terminal states back track assuming both player play optimally and pick the one with best reward for our player. Minimax is simple but it need too much storage to load the search tree and do the recursive calculation when the game has big branching factor.

Future improvement:

There are several possible ways for me to improve the Monte Carlo tree search I currently used for this project. First, I could try to use a different policy for default policy (The random rollout). I could try to do greedy simulation on both players till the end of the game if the running space is big enough. I could used different selection policy other than UCT. I could use MiniMax as selection policy which is to select nodes depends on the player turn. As mentioned in class, I could use Rapid Action Value Estimate to improve the algorithm. RAVE is

quote

“ - Assume the value of move is the same no mater when the move is played.

- This introduces a “bias” (simplification) in thinking but will reduce some variability in the Monte-Carlo estimates.

”

Reference :

[1] https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

[2] https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/

[3] L8-MonteCarloTreeSearch P36