


MCTS IMPLEMENTATION FOR GOMOKU

A PREPRINT

 **Jianzhi Shen**

School of Data Science
Fudan University
220 Handan Road, Shanghai 200433
20307130030@fudan.edu.cn

 **Keyu Mao**

School of Data Science
Fudan University
220 Handan Road, Shanghai 200433
20307130241@fudan.edu.cn

November 13, 2022

ABSTRACT

The project is centered at the implementation of MCTS for building a Gomoku agent. Apart from the explanation of the principle of MCTS, the advantage and disadvantage of MCTS are analysed and some optimization methods like root pruning or node pruning with heuristic sorting, simulation truncating by alternative terminate condition, are applied, and the effects are tested.

Keywords MCTS · Heuristic Application · Terminate Condition

1 Introduction

Throughout the past few decades, MCTS must be one of the most eye-catching search algorithms used in the game field. AlphaGo, which defeated Lee Sedol and Ke Jie in 2017, and AlphaZero, which dominated almost all chess games in 2018, both use MCTS as their core algorithm. The above examples are the products after some scholars combined MCTS with neural networks in 2016. Before 2016, MCTS had a long history of development. In fact, MCTS was proposed as early as the 1940s, and began to be widely used in the 1980s.

The Monte Carlo Tree search focus on using random sampling for deterministic problems which are difficult or impossible to solve using other approaches. In 2002, Chang et al. proposed the idea of "recursive rolling out and backtracking" with "adaptive" sampling choices in their Adaptive Multi-stage Sampling (AMS) algorithm for the model of Markov decision processes (1). AMS was the first work to explore the idea of UCB-based exploration and exploitation in constructing sampled/simulated (Monte Carlo) trees and was the main seed for UCT (Upper Confidence Trees).

In our experiment, we are expected to carry out MCTS method based on UCT and then apply this method to gomoku game to help agent find a proper move based on given board state. The report is simply divided into 4 parts:

- Introduction to basic Monte Carlo tree search
- Optimization of MCTS based on gomoku rules
- Experiment and results
- Conclusion and future work

2 MCTS

MCTS introduces randomness into tree search. The heuristic utility of a node is related to the empirical frequency of winning playouts of its successors. It is proven that MCTS is an estimate of Minimax(2), so it can handle huge branching factor without the help of an explicit utility evaluation. The search can come to a halt after reasonable rounds of simulation, so a time limit is introduced.

The iteration step consists of four parts: first it starts from the root, selects a successor by a certain criteria and repeats it on the sub-tree until it meets a leaf node; then identify the successors of the leaf node and expand it; then simulate from the successors to derive their values; after all these are done, backpropagate the values. The following are some details in the four procedures. The four procedures can be visualized in figure 1

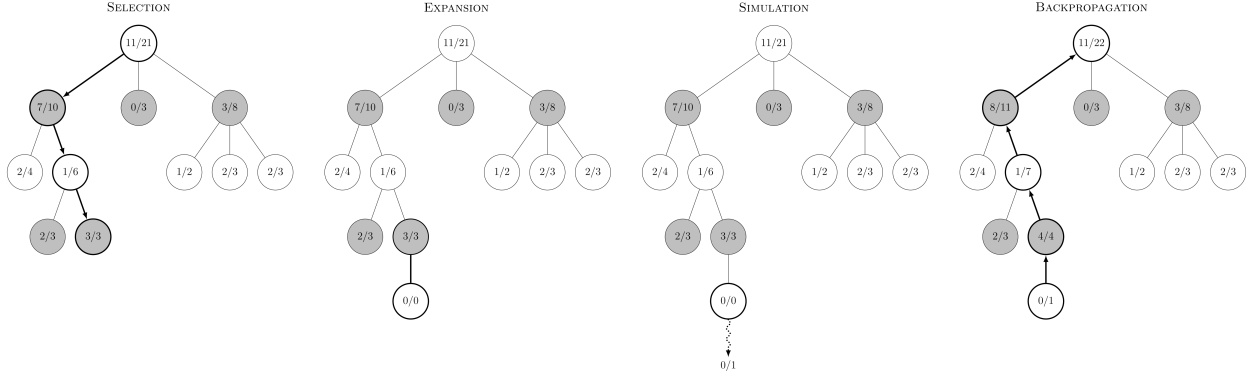


Figure 1: Four procedures of MCTS

2.1 Selection

At the beginning of search, the agent will start from root node and choose best child nodes of each layer until it finds a lead node (note that before expanding, even root node is a leaf node at first). The criteria of selecting best child nodes is UCT(3):

$$UCT = \frac{w_i}{n_i} + c \sqrt{\frac{\log N_i}{n_i}}$$

where

- w_i is the number of observed winning playouts from the action after the i -th move
- n_i is the total number of playouts from the action after the i -th move
- N_i is the total number of playouts from the root node of the action after the i -th move
- c is the weight of exploration

This criteria takes into consideration the balance between exploration and exploitation. If it only considers the frequency of wins, chances are that many nodes with great potential will not be visited after the first simulation leads to a lose. Now if a node has been visited too many times, it will leave the opportunity to its siblings.

2.2 Expansion

The expansion process identifies the successive actions and adds them to the tree. For a node which has not been expanded, all of its child nodes are equivalent since their private visited times are initialized as 0 so their UCT be explained as positive infinity. As for gomoku, it does not have to consider all the blank places. Only those near the placed pieces are worth considering. If any early threat is detected, see Section??, the successive actions can be limited to the corresponding vital moves. A local heuristic can be introduced to further reduce the branching factor, see Section 4.

2.3 Simulation

This process is aimed to simulate a series of game state so that the agent can in some degree determine which choice is most likely to bring it to a win state. To specify, this process completes a random playout from the newly found node. A playout may be choosing uniform random moves until the game is decided.

2.4 Backpropagation

After the simulation is done, the information can be propagated from the current node to the root. For example, if the playout is a win, n_i , w_i and N_i increase by 1 for all the nodes along the path. In backpropagation, all the nodes along the path will update the information concerning terminate result. And by using this updating method, we can ensure that the root node store the total visited times, which is of vital importance in computing UCT.

Algorithm 1 MCTS

Input: current state

- 1: Initiate $UCT \leftarrow 0$ for all nodes
- 2: $root \leftarrow \text{MAKE_NODE}(\text{current state})$
- 3: **while** within time and iteration limit **do**
- 4: $node \leftarrow \text{TREE_POLICY}(root)$
- 5: $action \leftarrow \text{POP}(node.actions)$
- 6: $state \leftarrow \text{MAKE_MOVE}(node.state, action)$
- 7: $new \leftarrow \text{MAKE_NODE}(state)$
- 8: $new.parent \leftarrow node$
- 9: $node.children.append(new)$
- 10: $result \leftarrow \text{SIMULATE}(state)$
- 11: **while** new is not null **do**
- 12: $\text{BACKPROPAGATE}(new, result)$
- 13: $new \leftarrow new.parent$
- 14: **Output:** pre-action of root's successor with maximum winning rate
- 15:
- 16: **function** $\text{TREE_POLICY}(node)$
- 17: **while** node is not terminal **and** $\text{IS_EMPTY}(node.actions)$ **do**
- 18: **if** $node.state.role = \text{ME}$ **then**
- 19: $node \leftarrow \text{node's successor with maximal UCT for ME}$
- 20: **else**
- 21: $node \leftarrow \text{node's successor with maximal UCT for OPPO}$
- 22: **return** node
- 23:
- 24: **function** $\text{SIMULATE}(state)$
- 25: **while** state is not terminal **do**
- 26: $action \leftarrow \text{RANDOM_CHOOSE}(\text{GET_ACTION}(state.role))$
- 27: $state \leftarrow \text{MAKE_MOVE}(state, action)$
- 28: **return** $\text{WINNER}(state)$
- 29:
- 30: **function** $\text{BACKPROPAGATE}(node, result)$
- 31: **while** node is not root **do**
- 32: $node.n \leftarrow node.n + 1$
- 33: **if** $result = \text{ME}$ **then**
- 34: $node.win \leftarrow node.win + 1$
- 35: **else if** $result = \text{DRAW}$ **then**
- 36: $node.win \leftarrow node.win + 0.5$
- 37: $node \leftarrow node.parent$

3 Advantages and Disadvantages

In the above sections, we have generally introduced the basic framework of Monte Carlo Tree Search. Actually, it has been proven that the evaluation of move in MCTS method will converge to Minimax(4). Nevertheless, the convergence needs some special conditions like a so called "Monte Carlo Perfect" game.

As for advantage, the feature distinguish MCTS from other algorithm is that the method doesn't need a specific value evaluation strategy. In Minimax searching with Alpha-Beta pruning, we have a construct a rather complex utility function which help us to determine the value of present board. Besides, Minimax search will depend on the effect of that utility function and if the function is not great enough to help agent analyze the board, the results of Minimax

algorithm will be terrible. According to last report, our method used in Minimax will iterate all the promising grids in every later can count the patterns in 4 directions, which is time-consuming and hard to comprehend. On the contrary, MCTS method is based on the key concept that use enough simulation to judge the best move. Simply implementing the game's mechanics is sufficient to explore the search space. As a consequence, Monte Carlo Tree Search can be employed in games without a developed theory. Moreover, as we let the agent focus on the child nodes with higher UCT, which is a criterion used to judge the promising degree, the tree expands asymmetrically. Hence the method is more efficient than some classic searching methods.

In terms of disadvantages, we won't focus on "trapped states" which is considered to be common especially when fighting against an expert(eg: AlphaGo's loss in its fourth game against Lee Sedol). The main problem in our experiment is time consumption. According to the principle of MCTS, the algorithm cannot achieve a decent results until it simulate sufficient times. However, in the experiment we have to take into account the trade-off between efficiency and intelligence. It's of vital importance to increase the simulating times within a give time consumption. We will consider a sequence of optimization methods to reduce the consumption in simulation process.

4 Optimization

4.1 Trials to terminate condition

In the MCTS method, we will roll out many times to find an appropriate node. So, we suppose that the rule of simulation will have vital influence on our agent, the simulation times in a certain time and the complexity of agent in particularly. In this section, we will discuss the trade-off between speed and complexity we consider during simulation process.

4.1.1 Original method

The first strategy is to terminate the game when five continuous chess pieces in same color appear on the board. This is the most classic and common choices of MCTS applied for gomoku and this method is actual a decent method since it take time consumption and complexity into account at same time. To be exactly, determine whether there is a continuous five on the board is a rather simple mission will not take too much time. Besides, when the agent find a continuous five, the game is end for sure. In that case, the value which is backpropagated from leaf node will surely connect to winning.

4.1.2 Live three search

Second, considering live three plays an essential role in a gomoku game since both our agent and opponent should react to live three immediately and a pair of live three will almost lead to win the game, we may also choose live three as our terminate condition. Nevertheless, this method have a couple of problems. On the one hand, if we simply terminate the game whenever we see live three, there is case that the simulation will end in the beginning when live three already exists on the board. The solution to this is quite simple. When live three already exists, the movement is fixed no matter the live three belongs to whom. If the live three belongs to us, we should construct a live four immediately. Otherwise, we should block the opponent live three to prevent scary live four. Hence, the agent should scan the board at first and take a fixed move if live three was found. Then, if live three doesn't exist, the simulation will begin. On the other hand, when applying this terminate condition, the agent will cling to live three. It's known to all that constructing live three is not the only method to win gomoku. For instance, 2 sleep four will also lead to a nearly absolute victory. As a consequence, the agent is expected to abandon some strategies in the game, so the complexity of our agent will drop in some degree.

4.1.3 Important patterns pre-recognition

Finally, we can use the same method as we used in Minimax search to figure out all the patterns which will almost lead to victory. We may not end the game only when the agent find continuous five. Thanks to the randomness of simulation process, when there are important chess patterns on the board such as live four and a pair of live three, there is small chance that the agent will choose the right move to end the game. This won't be a problem if we can simulate enough times as the empirical results will converge to interpret the value of significant patterns. But in the experiment, the time is limited so the agent will be likely to lose some significant patterns. Therefore, we can list some common important patterns and use them as our terminate condition. And just like the second method, the agent will scan the board and take a fixed move in some certain circumstance. In a word, the simulation process will devote to constructing impressive patterns, and the agent will end the game before the simulation part if impressive patterns already exist. This method increase the complexity greatly although the time consumption of simulation per time will increase. However, it's worth mentioning that when simulating times is large enough, this method will just worse than the first method. In that case,

when simulating for same times (large enough), the results between 2 methods will be almost same but the first method will take rather less time.

4.2 Heuristic Searching Application

In the Monte Carlo Tree Search method, the agent is expected to generate all possible actions with respect to present state. In gomoku, we will have to consider all the vacant grids and this procedure will become the main part of time consumption. Therefore, we should abandon some grids which are not likely to help the agent win the game. By dropping the volume of action list, the expanding process and simulation process can be accelerated a great deal.

First, as we have used in Alpha-Beta pruning, we will only choose grids that have ability or potential ability to win the game (in gomoku game, we will consider the patterns). As we all know, in a gomoku game, a player has to form a continuous five in a direction with same color to win the game. Hence, only grids that can help to construct patterns or block opponents patterns are meaningful. So first of we will only consider move between other grids that already have chess pieces. By only taking grids with neighbors into consideration, we can greatly decrease the actions in action list, and the agent can expand more adequately within given time.

Second, we will use a heuristic function to evaluate all the actions in the first layer (namely evaluate all the child nodes of root node). For a given state, the root node will probably possess more child nodes and the number of nodes in the first layer will severely influence the time-consumption and complexity of expanding process. If we can use some method to select most promising nodes among nodes in first layer, the whole searching tree will shrink several times. And by only querying promising nodes in layer 1, we can get a more well-expanded tree though we may abandon some rather good choices in first layer.

By recalling the principle of greedy pruning, we consider applying a utility function to all of child nodes of root node and select k nodes with high utility. Then, the root node will merely consider those k nodes as its child nodes. In the experiment, we need a splendid heuristic function that can greatly evaluate the value of the move. Due to that, we will consider the heuristic function we have used in Alpha-Beta pruning algorithm. As for a brief introduction, the heuristic function will mainly concentrate on the analysis of patterns formed by chess pieces. For each move in the action list, we first postulate that the move has been done. Then we consider patterns around the move in 4 directions. For each pattern, we give it an empirical score. Just add up the score of all patterns around a move we can get the heuristic value of it. This heuristic function works very well in our last experiment so we suppose it is also suitable for MCTS method to evaluate child nodes.

It's worth mentioning that through the heuristic application in the first layer, we can consider some situation with distinctive movement. For instance, when encountering "Live Four", "Five" of ours or "Five" of enemy, the movement is fixed. For the former one, the agent should choose to add a chess piece to the "Live four" can win the game. For the latter one, the agent should block the pattern or Sleep four will turn to a "Five" in the next turn for a wise enemy. When facing these 2 patterns, the agent will not do further search and take a move immediately, helping to reduce the time consumption. If we follow the principle of MCTS method, the agent will still simulate a great number of times and can't make choice unless time consumption reach limitation or simulation times are great enough. Such cases are quite common in a gomoku game, so this threatening patterns found in heuristic application process can help agent to make choice nearly immediately, especially in the final phase of the game (when the game is approaching a terminate state).

Also, we also try to apply the heuristic function not only for root node, but also the rest of node. In that case, we may abandon some potential win cases in every layer, but the tree can be expanded much deeper under time limitation. That means the results might be more reliable under the circumstance that we have already ignored some cases with lower probability to lead to win the game. Nevertheless, whether applying heuristic function only to root node or every node should be a trade-off in our experiment, we test and give the results in the next part.

5 Experiment and results

In this section, we are going to carry out a series of experiment. The type of experiment can be divided into 2 parts: test for optimization and test for simulation times.

5.1 Test for optimization

In the above optimization part, we propose 3 kinds of possible ways to ascertain the terminate condition. We have already explain their content and list their strengths and weaknesses theoretically. Hence, in this subsection, we will test their effect by making them fight against a sequence of AIs and compare their final results. (Note that time limitation exists)

Table 1: Results with different optimization methods

Heuristic	Terminate	fiverow	mushroom	pisq7	purerocky	sparkle	valkyrie	zetur17	scores
Only Root	Five	7:5	10:2	3:9	1:11	1:11	7:5	1:11	1027
Only Root	Threat Search	5:7	12:0	0:12	4:8	0:12	2:10	0:12	938
All Nodes	Five	9:3	9:3	1:11	8:4	0:12	3:9	1:11	1036
All Nodes	Live Three	6:6	9:3	2:10	2:10	0:12	5:7	0:12	953

¹ Heuristic: which nodes we will apply Heuristic function to(Only root or All nodes)

² Terminate: Terminate condition of simulation

³ Five:stop if a continuous five is found, Threat Search: stop if a threatening pattern is found, Live three: stop if a live three is found

The results are shown in table 1. We can find that first and third one performs well and all of above have higher score than mushroom(899).Threat search performs best when dealing with mushroom. Clearly, there is still great randomness in the test, and we should repeat the experiment to get further conclusion.

5.2 Simulation times exploration

In the game with time limitation, it's indispensable to set a proper time restriction per step for the agent. However, with fewer simulation times, the MCTS method will perform worse. Thus, we try to set max simulation times differently and get the competing results. In this subsection, we will list the game results against different AIs with different max simulating times and conclude which is the most appropriate choice.

Table 2: Results with different simulation times

Simulation times	fiverow	mushroom	pisq7	purerocky	sparkle	valkyrie	zetur17	scores
100	7:5	11:1	0:12	3:9	0:12	4:8	0:12	977
200(6s)	9:3	9:3	1:11	8:4	0:12	3:9	1:11	1036
200(8s)	5:7	10:2	4:8	6:6	1:11	2:10	0:12	1015

¹ 200(6s) means: max searching times is 200 and set time limitation as 6 seconds

Though the test has some randomness, there is a clear tendency that more searching times will lead to a better results. Time limitation has no obvious effect after 6 seconds per step.

6 Conclusion

In this project we successfully apply MCTS method to gomoku. Besides basic rules of MCTS, we also do some trials to optimize the method. for instance, we apply heuristic function to find appropriate actions. We test all the optimization method proposed in the report and explore the best terminate condition and simulation times.

Future work involves a further optimization method, which is based on the idea that the winning rate of some states can be stored and retrieved with the help of a Zobrist hash, and then the history winning rate can thus have some effect on current choosing policy, making the individual MCTS more informed.

References

- [1] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, "An adaptive sampling algorithm for solving markov decision processes," *Operations Research*, vol. 53, no. 1, pp. 126–139, 2005.
- [2] F. Chelaru and L. Ciortuz, "Combining old-fashioned computer go with monte carlo go," pp. 216 – 222, 10 2008.
- [3] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.
- [4] B. Bouzy and C. Tutorial, "Old-fashioned computer go vs monte-carlo go," in *IEEE Symposium on Computational Intelligence in Games (CIG)*, vol. 84, 2007.