1.5em 0pt

Minimax search α-β pruning algorithm for Gomoku

Cheng Zhong (16307110259)
Ruipu Luo (16307130247)
Course: Artificial Intelligence
DATE:12th January 2020

***Abstract*—In this project we using MCTS and Minimax search with α-β pruning algorithm for Gomoku. MCTS algorithm is based on Monte Karlo Tree simulation method, which goes through lots of simulation for current state and generate a game search tree. However, it is high space-complexity because it need to store all the pattern of the board and it's successor node. So, we finally choose the Minimax search - pruning algorithm for our AI. In order to save more time, we also added the concept of a threat space to directly find those high-threat points to save search time.**

***Index Terms*—MCTS, Minimax Search, α-β pruning, Threat space search, Gomoku**

I. Introduction

Nowadays, computer board games have been the focus of artificial intelligence search for a long time. In 1997, a computer named "Deep Blue" won world champion of chess Garry Kasparov. Also, "Alpha Go" made by Deepmind got 4 to 1 victory over Korean Go master Lee Sedol, after that, the upgraded version "Master" won a 60-game winning streak against professional Go players. In 2017, "Libratus" from CMU defeated Human players in no limit Texas hold'em.

Gomoku is a strategy board game of two players. Usually the two sides use black and white chess pieces respectively, and play at the intersection of the board's straight line and horizontal line. The one who first get continuous 5 chess in a line will win the game.

Gomoku has been proven that the one who play first will win. For fairness, in this project we will use a fixed start and exchange the first player to ensure that the winning rate of both sides is approximately equal.

II. Method

A. *MCTS*

Monte Carlo Tree Search is a method for finding the optimal decisions in a given state by simulating all the possible results and return the reward from the option to build the search tree. It's useful in GO games, according to DeepMind's paper[1], AlphaGo uses a novel method combining deep neural network with the Monte Carlo method to simulate the board and its best option.

According to C.B.Browne and his collaborators work[2], MCTS requires a large number of simulation to bulid up a large simulation tree in order to get more accurate to its reward. Also, it divided MCTS into four parts: Selection, Expansion, Simulation and Back-propagation. At the Selection

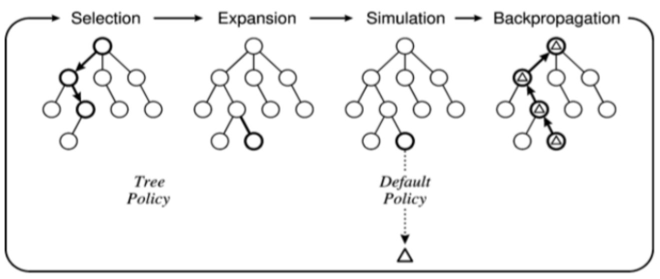


Fig. 1. The basic process of MCTS

parts, the tree begins from the root node and recursively applies the Tree Policy to descend through the tree until it reaches the leave node. And then the Expansion parts will add child nodes to expand the tree according to its heuristic knowledge or any possible choices. In simulation parts, we run a simulation from the expand node to judging the possible benefits of this decision. Finally, we use the Back-propagation to select the child node with the best outcome.

In this project, we implemented a MCTS structure named HMCTS mentioned in Zhentao Tang and his collaborators' work[3]. In this structure, if the board's feature follows any of the heuristic knowledge, then we execute the unique option from the knowledge, or we will randomly select a option from its possible move. The algorithm is presented in Figure.2. Noted that the heuristic knowledge is the common knowledge for players like we should block opposite player's four-in-a-row and to emerge five in a row in my side when possible. Using the knowledge can save time in simulation than random sampling and getting earlier converge. In simple terms, it is like a simplified version of thread space search we will show in III.A.

However, we found that for a 20 * 20 chessboard, the search space for each step in MCTS is too large to return results in a specified time. Meanwhile, because it used a tree structure to store the feature of the chessboard, the same features will be repeatedly calculated, which greatly reduces the calculation efficiency. Maybe the algorithm will perform better when we add more complex knowledge in high level

```
input original state $s_0$;
output action $a$ corresponding to the highest value of MCTS;
add Heuristic Knowledge;
obtain possible action moves $M$ from state $s_0$;
for each move $m$ in moves $M$ do
  reward $r_{total} \leftarrow 0$;
  while simulation times < assigned times do
    reward $r \leftarrow$ Simulation($s(m)$);
    $r_{total} \leftarrow r_{total} + r$;
    simulation times add one;
  end while
  add ($m$, $r_{total}$) into $data$;
  end for each
return action Best($data$)

Simulation(state $s_t$)
  if ($s_t$ is win and $s_t$ is terminal)  then return 1.0;
                                           else return 0.0;
  end if
  if ($s_t$ satisfied with Heuristic Knowledge)
    then obtain forced action $a_f$;
         new state $s_{t+1} \leftarrow f(s_t, a_f)$;
    else choose random action $a_r \in$ untried actions;
         new state $s_{t+1} \leftarrow f(s_t, a_r)$;
  end if
  return Simulation($s_{t+1}$)

Best($data$)
  return action $a$   //the maximum $r_{total}$ of $m$ from data
```

Fig. 2. The Algorithm of MCTS

and use a graph structure to store the feature.

B. *Minimax search with α-β pruning*

*1) **Algorithm implementation**:* After studying the sample code carefully, the strategy we need to fill the **brain turn function**. The effect of function is that input a current state and return the optimal move of next state.We naturally think of using the Minimax search algorithm in the adversarial search method to solve this problem. However, because there are too many backgammon chess board states, if only the Minimax search is used, it will time out and cause losing the game. So we use the method of Alpha Beta pruning to save calculation time. The Pseudo code of Minimax is shown below.

The main idea of Minimax search algorithm is to simulate that the opponent is a savvy person playing Gomoku with you, so when we recursively predict the score of a point, we need to alternately calculate the maximum and minimum values. And return the score and position for each choice.An illustration

```
function ALPHA-BETA-SEARCH($state$) returns an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value $v$

function MAX-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow -\infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha$, $\beta$))
    if $v \geq \beta$ then return $v$
    $\alpha \leftarrow$ MAX($\alpha$, $v$)
  return $v$

function MIN-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
  if TERMINAL-TEST($state$) then return UTILITY($state$)
  $v \leftarrow +\infty$
  for each $a$ in ACTIONS($state$) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$) , $\alpha$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
  return $v$
```

Fig. 3. The pseudo code of Alpha-Beta-Pruning

for Minimax search algorithm is shown in fig.4.

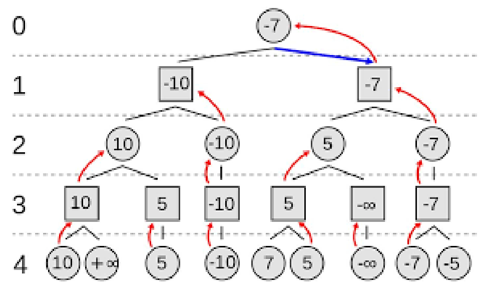
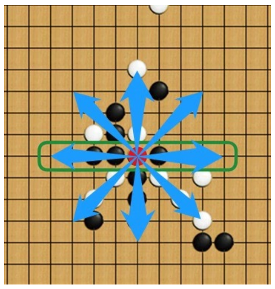


Fig. 4. An simple illustration of Minimax search

*2) **Evalution**:* No matter which method we use, we must design an effective heuristic to evaluate Gomoku state. The evaluation function has a wealth of Gomoku knowledge, which can detect the precise details of the wooden board and therefore can properly guide our search algorithm.

We use points on board as an element of the evaluation process. According to Gomoku rules, we check the four directions of a point (shown in fig.5(a)), if a line satisfy a certain pattern then score the line, the score of each pattern is shown in fig.5(b). And finally sum the scores of the four directions as the point's score.

Unfortunately, because movings are very frequent in our searching processes, it can be extremely slow if we rescore all the points on the board in every direction. To speed up the score update process, we use two strategies here. First, since Gomoku plays chess locally (usually 6 6 blocks), we assume that there are new actions on the board that affect its neighbors. Second, We only up- date score of neighbor points in one direction, other than all the four directions.


(a) Direction

Pattern	Score
-xo-	1
-o-	10
-xoo-	10
-oo-	100
-xooo-	100
-ooo-	1000
-xoooo-	10000
-oooo-	100000
ooooo	10000000

(b) Score

Fig. 5. the 4 direction of a point and pattern score

III. Optimization Methods

A. *Threat Space*

In some states, if we can't make the right move, the opponent can win by constantly making the blocked-Four[4] or other features that you can only make fixed moves to answer it. Similarly, we can also win in this way. However, because of its limited search depth, minimax algorithm cannot obtain such correct results. In Figure 4, the white one can win when place on the location marked with a red circle. However, if using minimax algorithm in a shallow depth, we may not find this place.

Inspire by this, before we run a minimax search with α-β pruning, We will search some pre-determined threat patterns. When the same pattern is found, we will directly output the solution without going through α-β pruning based on the heuristic knowledge which we have input. This will not only improve the efficiency but also ensure the accuracy of the results.

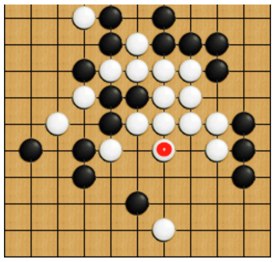


Fig. 6. The thread patterns

B. *Candidate Sorting*

Our strategy for expanding the nodes of each layer of the search tree is to find empty points with a distance of 1 around each point. After reading the textbook,we recognize that the performance of alpha-beta pruning is greatly infected by the order of search nodes. For example, a max-node should acquire the maxvalue from candidates as soon as possible. If we sort all candidates properly, the pruning speed can be improved greatly. So, we sort all candidates according to its point score described in second Section.

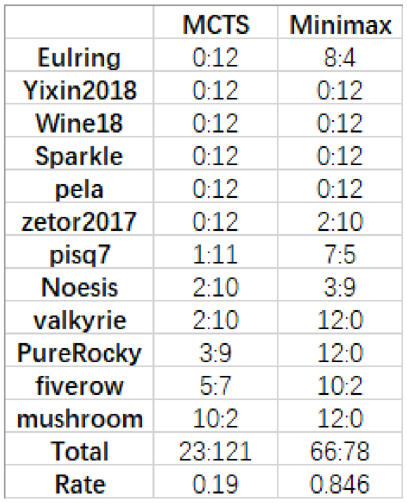
C. *Greedy Pruning*

Alpha-beta pruning is a safe pruning method in Minimax search, in other words, it will definitely find the best results at a fixed depth. We want to relax it to a sub-optimal version in order to find approximate "best" results to increase search depth. We use a simple but effective method called greedy pruning to increase search depth. In Section II, we introduced candidate pruning (safety) and candidate ranking by point value. We know that the best move for a player or opponent is probably the player with the higher score. In "greedy pruning", we keep only the top b candidates in each layer, thus exploring higher depths at the same time.

IV. EXPERIMENT

In the final match, three start modes were fixed. The time of each step of each agent should not exceed fifteen seconds, and the total game time should not exceed 90 seconds.

We used the optimized Alpha-Beta pruning minimax search agent and MCTS agent to play against with the given 12 agents under the given three starts. The results are as Figure.7. From the figure, we can find that the pruning algorithm can reach the level of pisq7, and the simple MCTS can reach about fiverow.



	MCTS	Minimax
Eulring	0:12	8:4
Yixin2018	0:12	0:12
Wine18	0:12	0:12
Sparkle	0:12	0:12
pela	0:12	0:12
zetor2017	0:12	2:10
pisq7	1:11	7:5
Noesis	2:10	3:9
valkyrie	2:10	12:0
PureRocky	3:9	12:0
fiverow	5:7	10:2
mushroom	10:2	12:0
Total	23:121	66:78
Rate	0.19	0.846

Fig. 7. Result of Gomoku Championship

V. Conclusion and Further improvement:

In this report, we implemented MinimaxSearch, which introduced a point-by-point strategy to build an effective evaluation system, and adapted candidate pruning threat space and greedy pruning to increase the tree's search depth. As a result, we can get some conclusion below

- The pruning algorithm can reach the chess power of pisq7, and the simple MCTS can reach about fiverow
- We can reduce the search time each step to increase the search depth and impove the effect of improving chess power

- For the MCTS algorithm, we can try to cache the simulation result using the Zobrist Algorithm to improve the efficiency.
- Also, we can using the Q-learning and TD method to improve the MCTS algorithm, like UCT in Zhentao Tang's work[3].
- For minimax search - pruning algorithm, the further improvement is to add more thread patterns and improve the generalization ability of the model in different situations.

REFERENCES

[1] L. V. Allis, H. J. Herik, and M. P. H. Huntjens. Go-moku and threat-space search. 1993.

[2] C. B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence Ai in Games*, 4(1):1–43.

[3] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[4] Zhentao Tang, Dongbin Zhao, Kun Shao, and Le Lv. Adp with mcts algorithm for gomoku. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2016.

[3] [2] [4] [1]