Final Project Gomoku

Ke Fan 17307110287 Siyu Yuan 17307110448

Abstract

Gomoku is a zero-sum game. In this project, we first tried the MCTS algorithm, but then give up due to the unsatisfactory experimental results. To get a better AI, we turned back to the Minimax search tree with limited depth combining alpha-beta pruning. To estimate the utility of non-termination state, we bring in some dominate compositions with score, and then make a linear weighted combination of the matching dominant compositions to determine these non-termination states' utility. Besides, Zobrist table and local search with update is used to speed up the search.

Keywords: Minimax, Alpha-beta pruning, Utility, Zobrist, Local update, Local search

1 Introduction

In the recent years, the artificial intelligence used in the game field has a relatively mature development. Game AI overcomes many complicated games with varied solutions, such as tic-tac-toe, chess, and go. Gomoku is a popular two-player strategy board game. Traditionally, Gomoku is played with pieces (black and white) on a board with $15x15$ intersections. The winner is the player who first obtains a complete row of five pieces from a horizontal, vertical, or diagonal direction. Monte Carlo Tree Search and Minimax Search Tree are the most classic Gomoku algorithms. However, as William says[1], it is impossible to complete a search completely.

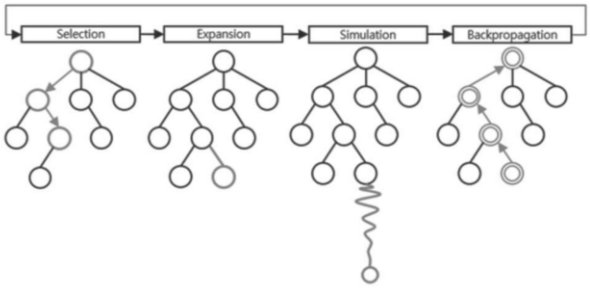
Fortunately, alpha-beta pruning can be used to speed up Minimax search tree[2]. In this article, we will use the Minimax search tree with limited depth combining alpha-beta pruning to complete the initial algorithm of the Gomoku game. In the meantime, we also introduce the Zobrist table and local search with update to speed up algorithms.

2 Method exploration

2.1 MCTS

Monte Carlo Tree Search is a search technology in the field of artificial intelligence. From a global perspective, the main goal for MCTS is to choose the best next step according to the given state.

This search algorithm mainly includes four stages, that is, selection, expansion, simulationand backpropagation.



Search is a collection of traversals along the game tree. A single traversal is the path from the

[1]William Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders[J]. Journal of Finance, 16(1):8-37.q

[2]Knuth D E , Moore R W . An analysis of alpha-beta pruning[J]. Artificial Intelligence, 1975, 6(4):293-326.

root node (the current game state) to an incompletely expanded node. When encountering a node that is not fully expanded, select a child node that has not been visited for a simulation. The results of the simulation are then backpropagated to update the ancenstors' value of the current tree. When the search ends, you can decide what to do next based on the prior Information.

However, after implementing MCTS, we found that the behavior of our AI is far less intelligent than before. In some simulations, it cannot foresee some intentionally designed actions, so it is easy to be defeated by opponents. After our analysis, there are two reasons. On the one hand, due to time constraints, we cannot simulate all the movements, so we cannot accurately evaluate the board. On the other hand, MCTS lets AI take action at will, and it is difficult to use effective heuristics to guide simulation. Therefore, we abandoned the MCTS algorithm and turned back to optimize the Minimax search tree with alpha-beta pruning.

2.2 Minimax

Minimax is a decision rule for artificial intelligence, decision theory, game theory, statistics, and philosophy. It is used to minimize possible losses in the worst case. When dealing with profit, it is called "maximizing" , that is, maximizing the minimum profit[3].

2.3 Alpha-Beta Pruning

The algorithm maintains two values, alpha and beta, which represent the minimum value guaranteed by the maximizer and the maximum value guaranteed by the minimizer. At the beginning, alpha is minus infinity and beta is plus infinity, that is, both players start with their worst score. When the maximum score obtained by the minimizer (i.e. "beta" player) is less than the minimum score obtained by the maximizer (i.e. "alpha" player), the maximizer does not need to consider the subsequent nodes of this node, as they will never be reached in the actual game[4].

2.4 Zobrist Hash

Zobrist hash is used in computer programs that play abstract board games to implement transposition tables (a special hash table), indexed by board positions, and used to avoid analyzing the same position multiple times.[5] It is widely used due to its good compatibility. The construction of Zobrist Hash data structure will be introduced in detail in Part Three.

3 Algorithm Introduction

As we know, Gomoku is a zero-sum game. Therefore, Minimax search tree can be apply to implement the AI for this game.

However, due to the large search space and limited computing resources, we cannot extend the full minimax search tree in Gomoku. Therefore, we decided to explore only limited depths, that is, we used a minimax search tree with limited depths. Since we limit the depth of the search tree, the state of the deepest node are not terminal state, so we have to design a good evaluation function to estimate the utility of non-terminal state.

3.1 Board Estimation

From the above analysis, we know that the key for Minimax search tree with limited depth is the evaluation function for the non-terminal states.

[3]M.E. Johnson, L.M. Moore, D. Ylvisaker. Minimax and Maximin Distance Designs[J]. Journal of Statistical Planning Inference, 1990, 26(2):131-148.

[4]Knuth D E , Moore R W . An analysis of alpha-beta pruning[J]. Artificial Intelligence, 1975, 6(4):293-326.

[5]Albert Lindsey Zobrist, A New Hashing Method with Application for Game Playing, Tech. Rep. 88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, (1969).

Due to the time constraint and our lack of knowledge about Gomoku, we borrowed ideas from previous generations[6], namely,
(1) List the types of composition that are dominant to our AI and convert these composition into strings, while attaching a score to each type of composition.
(2) In the process of the game, loop through the rows, column and diagonals, and use regular expressions to match the dominant composition.
(3) make a linear weighted combination of the matching dominant composition to determine the next action of our AI.

3.1.1 dominant composition

According to the relevant information, we obtained the basic Gomoku composition.

The "4" is the first step to the victory of Gomoku. The "4" here refers to the single chess form that can be formed by another move of the party, including "H4" and "C4". "H3" (the figure is below) is the most common type of attack. After "H3", if the opponent does not pay attention, we will be able to turn "H3" into "H4" in the next hand, and we know that "H4" is not simply defensive. "M3" can only form "C4". Compared "H3", the risk of "M3" is reduced a lot, because for the "M3", even if not to defend, the next hand can only form "C4". For the simple "C4" type, as we mentioned above, it can be defended. "H2" is not blocked by the two connected pieces, which can form "H3", while "M2" can form the "M3".

Based on the above description of the dominant composition, we specify the score about them.

composition	score
WIN	2000000
"H4"	10000
"C4"	500
"H3"	200
"M3"	50
"H2"	5
"M2"	3

3.1.2 Convert composition to String

In the broad, We can convert the dominant composition mentioned above to a string that can be matched by a regular expression, where "1" is our AI's piece, "2" is opponent's piece, and "0" is no piece. For example, "H4" can be expressed as "011110", C4 can be expressed as "211110" and so on.

3.1.3 Weight of the dominant compositions

We have converted the dominant compositions into strings and specified score. Now we only need to traverse the target board to find the corresponding dominant composition, and make a linear weighted combination to gain the utility of the non-terminal state.

Intuitively, we should set the weight of every dominant composition to 1, since we've designed each dominant game to have a different score. However, taking into account that when the opponent's composition is "H4" or "C4" and we don't have, we should be inclined to stop the opponent from wining rather than pursue our own victory. Therefore, when the opponent's composition is "H4", "C4" and "H3", the weight should be increased.

[6] zjh776. Intelligent gobang based on the algorithm of Alpha-Beta Pruning. https://www.iteye.com/blog/zjh776-1979748, 2018-01.

composition	our weight	opponent's weight
WIN	1	10
"H4"	1	10
"C4"	1	10
"H3"	1	10
"M3"	1	1
"H2"	1	1
"M2"	1	1
"S4"	1	1
"S3"	1	1
"S2"	1	1

3.2 Minimax and alpha-beta pruning

As we mentioned aboved, we need to construct Minimax search tree with limited depth. Then we can easily adopt alpha-beta pruning onto minimax search tree. The Pseudocode is as follows:

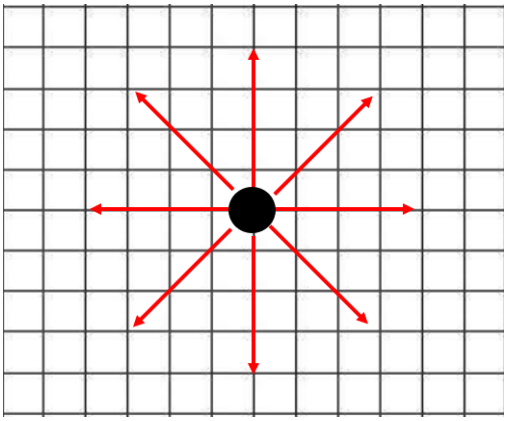
```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

3.2.1 Local Search

In order to ensure our AI play within 15 seconds, we set the minimax search depth at 1 levels and restrict search space within 8 directions centered on on successor's action.



Through local search, we can obtain two local values before and after the occurrence of a specific action, and greatly reduce the number of branching factors.

However, though we just consider local search, in the beginning of the game, the branching factors are still so large that our AI cannot move in chess within 15 second. To deal with this conundrum, we first calculate the utility for each branching factor in the first layer, and select the nodes for the states corresponding to the top four utilities. That is, in our turn, we need to find the first four highest utilities, while in opponent's turn, we need to find the first four lowest utilities. After that we only expand these nodes.

3.2.2 Local Update

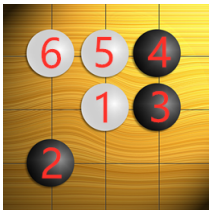
Inspired by local search, we can evaluate the board through local updates. According to the method of board evaluation in 3.1 section above, whenever there is a change on the board, we will loop through the all rows, column and diagonals. In fact, if an empty spot is placed on the board, the score will change only in the eight directions centered on successor' s action. Therefore, when we update the score, we only need to perform partial updates, which can greatly reduce the amount of calculation. The local update formula is as follows,

$$difference = V(local^{'}) - V(local)$$

$$V_{new}(global) = V_{old}(global) + difference$$

3.2.3 Zobrist Hash

We have already discussed the method to board evaluation. However, in the process of evaluating the board, we actually performed a lot of repeated operations. Just like this:



if we come across the above two cases in the search, we will evaluate both cases once, but in fact, with the first score, we can cache it, and then we can use the cached data instead of scoring it the second time. In fact, it is different from the above two cases only in order, and the final situation is the same. For most Gomoku games, it doesn't matter how you got there, as long as the pieces are the same, the situation is the same.

So now the question is, how do we represent a situation? Obviously, we need to build a hash table. This hash table must meet the following two requirements,
(1) The hash table must be able to represent the score corresponding to each game.
(2) The conflict of the hash table must be small enough to minimize the error.

Zobrist is a fast array hash algorithm that meets our needs. For the Zobrist hash algorithm, Only one xor operation is required for each move, and the performance cost of this xor operation is negligible.

Zobrist is implemented as follows:

1 Zobrist Hash Table

```
hashtable: Zobrist Hash Table, dic: A dictionary of cached random numbers
function INIT_ZOBRIST(length, width)
    dic ← {}
    hashtable ← {}
    D is a two-dimensional array
    for i = 1 → length do
        for j = 1 → width do
            dic[(i, j)] ← [random.randint(1, 2^64), random.randint(1, 2^64), random.randint(1, 2^64)]
        end for
    end for
    for i = 1 → length do
        for j = 1 → width do
            D[i][j] = self.dic[(i, j)][3]
        end for
    end for
    key ← 0
    for i = 1 → length do
        for j = 1 → width do
            key ← key XOR D[i][j]
        end for
    end for
    hash_table[key] ← 0
    return hash_table
end function

function COMPUTE(hashtable, position, player, old_key, value)
    get the random number num for the dic according to the position and player
    new_key = old_key XOR old_num XOR new_num
    hash_table[key] ← value
    return hash_table
end function
```

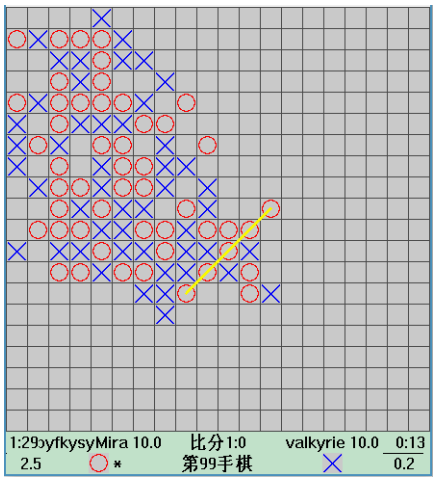
In the meanwhile, it is obvious that if the Zobrist Hash value of the current board has been stored in the Zobrist Hash, the current board has been expanded, so it can be directly pruned. Therefore, the Zobrist algorithm can also play a role in pruning.

4 Result

In this project, we implement the above algorithm.

limited depth	branch factor
2	10

Wonderfully, we get the 1242 points. One victory is as follows,



5 Reference

Albert Lindsey Zobrist, A New Hashing Method with Application for Game Playing, Tech. Rep. 88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, (1969).

Knuth D E , Moore R W . An analysis of alpha-beta pruning[J]. Artificial Intelligence, 1975, 6(4):293-326.

M.E. Johnson, L.M. Moore, D. Ylvisaker. Minimax and Maximin Distance Designs[J]. Journal of Statistical Planning Inference, 1990, 26(2):131-148.

William Vickrey. Counterspeculation, Auctions, and Competitive Sealed Tenders[J]. Journal of Finance, 16(1):8-37.

zjh776. Intelligent gobang based on the algorithm of Alpha-Beta Pruning. https://www.iteye.com/blog/zjh776-1979748, 2018-01.