Project: Gomoku Game (15x15)

Type of Project: Game Tree Search

Bin Liu – binliu:

Heuristic function(Major) Experimental assessment(Minor)

GUI(Major) Min-Max-Cut Algorithm(Major)

Bingzhao Shan – shanbin1:

Problem Encoding(Major) Experimental assessment(Minor)

Game Tree construction(Major) Manuscript(Minor)

Project Motivation/Background

What the problem is?

The game we choose is called Gomoku. Gomoku is an abstract strategy board game that has different version. In our version, the game contains a board with 15X15 intersections. It has two players. Player 1 holds Black and plays first while player 2 holds White, and players alternate in placing a stone of their color on an empty intersection. The winner is the first player to get an unbroken row of five stones horizontally, vertically, or diagonally.

Approach to the problem?

We utilized game tree search using minimax algorithm on this problem. In addition, we provided depth attribute so we can adjust how deep we will go.

Why do we choose game tree?

Gomoku fulfill all the game properties:

1. It has two players
2. Its state can be mapped on discrete value. If the game is an end state, we can use 1 represent player 1 win,-1 represent player 2 win. 0 if it is not an end state
3. Its state is finite. (Even though the total states in a 15X15 game is really huge)
4. It is zero-sum, which means if an player wins another player must lose
5. It is deterministic, there is not chance involved
6. It fulfills perfect information, because the board is seeable by both players

Thus, we can use knowledge we learned such as minimax algorithm on game tree search to solve this problem

Methods

How did we formulate our problem and what algorithms did we employ to solve it?

This problem is encoded as a Gomoku state, and we also encode a AI class. AI_class has its search method. In short, a problem can be given to a AI and AI can return a new state after AI has made his decision and added his stone into board. Player can also add stone into a given state. Depend on whose turn, AI and Player alternate in placing stone of their color and position on the current state, and result new current state. After each new state is produced, we would check is the state has a winner and show that on GUI.

We employed minimax algorithms with alph beta cut to solve this problem, since the states are too many, we use a depth attribute to limited the tree depth.

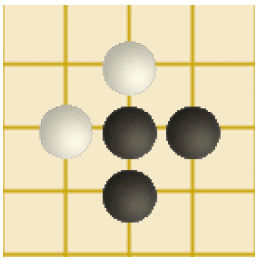
How do we encoding our state?

As we've mentioned before, that state space of this game is incredible large, thus we decide to use depth limited Game-Tree-Search on this puzzle.

1.GameState:

For each configuration of the board, we make a state variable for it. The State variable remembers some useful information we'll use in the future of this game.

To demonstrate the attributes of the state variable, we are now using a board of size 5x5 with player1 using black chess, player 1 play first:



State s:

```
s.board = [[0,0,0,0,0],
           [0,0,2,0,0],
           [0,2,1,1,0],     # the board configuration
           [0,0,1,0,0],
           [0,0,0,0,0]]
s.last_stone = (2,2)        # the last chess that is placed on the board
s.turn = 2                  # now it's Player2's turn to make next move
```

2. Successor Function:

We use two type of successor function in our project, one is used to get all possible move made by the player, the other is used to generate all good move of the AI itself.

To demonstrate, we choose a board of size 3x3,

with "*" represent the empty spot,

"X" represent the black chess used by player1

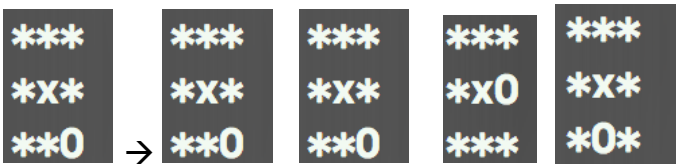
"O" represent the white chess used by player2

(1) The first Successor Function is called Successor:

It will return all possible moves made by another player:

(2) The Second successor function is called reduced successor:

It will return the moves made by another player, which is connected to some chess on the current board:



3. Heuristic Function:

Since we use depth limited Game Tree Search, it's important to calculate the heuristic of the leaves properly. Our heuristic function will analyze all the chess on the board, detect the maximum connection number it is connected to chesses with the same color, and return the

heuristic value based on the turn. Large Heuristic value of some state means that they are more likely to lead success for this player.

4. Goal_function:

The Gomoku_goal_function, named as detect winner in our project, detects whether there is some player who gets an unbroken row or five stones horizontally, vertically or diagonally.

Evaluation and Results

Evaluation1: Compare the response time for a Game-Tree-Search without alpha-beta cut and a Game Tree-Search with alpha-beta cut, depth=3 for both cases.

Make some number of move on the board, moves are the same in both two cases:

	GTS Without Alpha-Beta cut		GTS With Alpha-Beta Cut	
Number of move On board	Total Response time /second	Total Node explored	Total Response time /second	Total Node explored
3	3.71	180829	2.16	39721
5	31.68	501922	6.81	113389
7	49.38	1254389	10.12	256248

Conclusion : Alpha-Beta Cut for a Game-Search Tree with depth >= 3, saves a lot time!

Evaluation2: Compare on different depth of the Game-Tree-Search with alpha-beta-cut

Make 5 moves each case, moves are the save.

Depth	Total response time /second	Total Node explored
1	0.01	149
2	1.22	33732
3	6.2	122787

Conclusion : The Game-Tree-Search tend to make “wiser” choice as the depth increases, but the response time grows at a very fast speed.

Evaluation3: Response time and Node explored as we play, using Game-Tree-Search with alpha-beta-cut and depth = 3.

Start from an empty board and play step by step,

Number of Move on Board	Total response time /second	Total Node explored
1	0.53	12923
2	0.71	20617
3	0.93	35920
4	2.03	47146
5	1.91	64722
6	3.17	68739
7	5.23	101120

Conclusion: The response time of the AI slows down as game goes on.

Limitations/Obstacles

The largest barrier we encountered is about the search space of the board. Since it is a 15*15 board and each intersection can have no stone, black stone or white stone. There could be 3 to the power of 225 states for the game state. It is not possible for computer to explore that much code. So we have to limited states we search and give a result at reasonable time. The approaches we used are:1. limit the depth of search function.2 Reduce the successors of each state by delete the obvious node that does not has best outcome. By doing that, our function return a one of the best result in reasonable time. (More depth cost more time but give better result).

Conclusions

In this project, we have shown how to utilize game tree search on Gomoku problem and what the limitation is. We also test the actual performance between minimax search with ab cut and without ab cut. In addition, we have shown how the depth of search could affect the run time of algorithm. We are surprised how alpha-beta could dramatically reduce the search space and improve the efficiency of minimax algorithm.

For further reduce the time cost of minimax, we could try to make two improvement. 1. To find a better heuristic function 2. To further reduce the successor of each state (by deleting some states that are not obvious but surely will not give the best outcome).