**GOMOKU ALGORITHM STUDY**

**MIN-MAX AND MONTE CARLO APPROACHING**

**Author: Jingtong Liu, Wei Sun, Weixun Ge, Guochen Xie**

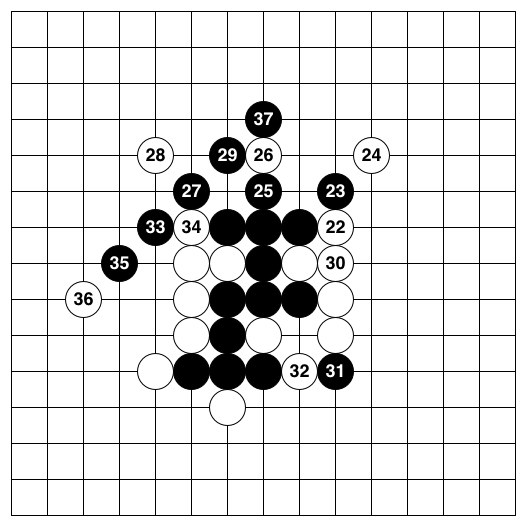
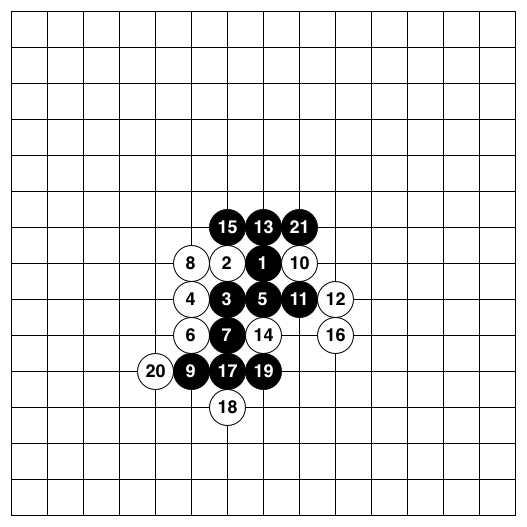
**Abstract:** We will implement two playing agents for the popular board game Gomoku. Min-Max Search Tree algorithm and pattern recognition is used for one of the agents, which is a classic algorithm for Gomoku. And another one, we attempt to find the acceptable move for next step by the combination of part of Min-Max search Tree, Genetic Algorithm, Monte Carlo search Tree. By using the second approaching, we try to find an acceptable solution, with which we need not search all deep down the Min-max Tree. That approaching has high research value when the searching space is quite big or unlimited. With comparing and contrasting the preferment of two agents, some interesting phenomenon has been found, which we attempt to explain that with our limited knowledge background both in Computer Science and Gomoku.

**Keywords:**Gomoku, Min-Max, Monte Carlo, Genetic Algorithm**,** Modification.

**INTRODUCTION**

* **About the Game**

**Gomoku** is one of the most popular board games. Two players will play alternatively on a 15\*15 boards. Moves are only allowed in the intersections .And to win the game, one has to let his 5 stones connected in a horizontal, vertical, or diagonal line.



**Pic1: sample of the winning cases of the black stones**

* **About the Algorithm**

Recently the study on Gomoku is focused on the pattern recognition combining with the Min-Max decision tree, which is mostly relay on the domain knowledge of the Gomoku experts. The complex of the decision tree is , and any four-step forecast need room of . As a result, how to optimize algorithm has become a challenging task. However, the more we want to cut off from the decision tree, the more complex our evaluation function will be, which is the heel of Achilles, too.

What is more, we try to do something different from the classic approach, even though it might not be as good as the traditional one. We are wondering if it is possible to use no special knowledge as all! Suppose there are two idiots playing this game without any knowledge of Gomoku, except the rules. The only way for them to win is learning from their own experience. They keep playing that. After thousands of times, they may be a quite strong player.

Without including any Gomoku Knowledge beyond the rule, we believe an implementation using Monte Carlo algorithm can also achieve strong level of Gomoku. There are two assumptions: First, the better move will be played in most of the winning games. Second, two idiots can play the game thousands times.

First**,** we introduce Min-Max approaching. **Min-Max** is a decision rule used in decision theory, game theory, statistics and philosophy for minimizing the possible loss for a worst case (maximum loss) scenario. Alternatively, it can be thought of as maximizing the minimum gain (maxmin). Originally formulated for two-player zero-sum game theory, covering both the cases where players take alternate moves and those where they make simultaneous moves, it has also been extended to more complex games and to general decision making in the presence of uncertainty.



**Pic 3: a classic Min-Max Search Tree**

**Monte Carlo** is a class of computational algorithms that rely on repeated random sampling to compute their results. Monte Carlo methods are often used in computer simulations of physical and mathematical systems. These methods are most suited to calculation by a computer and tend to be used when it is infeasible to compute an exact result with a deterministic algorithm. This method is also used to complement theoretical derivations.

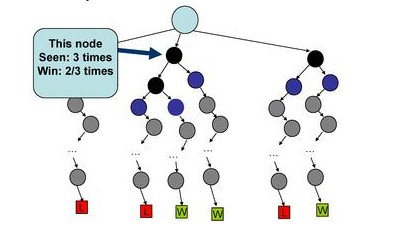
**Monte Carlo Search Tree:**

Here is our proposal:

(1) Moves are performed randomly with the probabilities assigned by the method of simulated annealing,

(2) The value of a position is defined by the win rate of the given position and the frequency that the move has been played

(3) To find the best move in a given position, play the game to the very end as suggested by (1) and then evaluate as in (2); play thousands times such random games, and the best move will be the one which does best.



**Pic 4: Monte Carlo Gomoku**

**A Dynamic Bayesian Network (DBN)** is a Bayesian Network which relates variables to each other over adjacent time steps. This is often called a Two-Time slice BN because it says that at any point in time T, the value of a variable can be calculated from the internal repressors and the immediate prior value (time T-1). DBNs are common in robotics and have shown potential for a wide range of data mining applications.

**Pic 5: example of a DBN.**

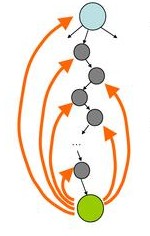
**Genetic algorithm:** In the computer science field of artificial intelligence, a genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.

**TECHNICAL APPROACH**

**NAÏVE THOUGHT OF THE TWO APPROACHES**

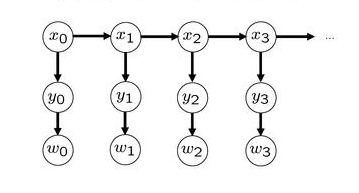
**Monte Carlo Forward Approach:**

1. In each step of simulate annealing, update the values of all positions played in rollout, store the value (computed by proposal (2) ) for each node.



**Pic 6: update values of all positions in rollout**

(ii) Given the value of the nodes, find the 5 best moves of the nodes, play them randomly, and then find the move should be performed. (According to proposal (1)) . Record the values of the nodes computed by (i).



**Pic 7: Bayesian Network**

(iii) After the opponent agent played one move, continue the Step (ii) till the game end.

**Min-Max Backward Approach:**

Min-Max:

**Function** integer minimax(node, depth)

**If** node is a terminal node or depth <= 0:

**Return** the **Evaluation Function**( node )



**For** child in node: # evaluation is identical for both players

α = **max**(α, -minimax(child, depth-1))

**Return** α

**Evaluation Function:** Pattern Recognition assign scores to different patterns

A move has two type of values: attack value (means playing that move can form a "good" pattern) and defense value (means playing that move can destroy opponent's "good" pattern ). Then we give a definition of "good" shape. And try to assign value to each of them.

Patterns are as follow:

|  |  |  |
| --- | --- | --- |
| C:\Users\sunwei\Desktop\double-3.jpg | C:\Users\sunwei\Desktop\xxxx.jpg | 说明: C:\Users\sunwei\Desktop\oxxxx.jpg |
| **double three value() = 9970** | **four-in-row value() = 9990** | **straight-four value() = 9980** |
| C:\Users\sunwei\Desktop\xxx.jpg | C:\Users\sunwei\Desktop\oxxx.jpg | C:\Users\sunwei\Desktop\double-4.jpg |
| **three-in-row value() = 200** | **straight three value() = 10** | **double four value() = 9998** |
|  |  |  |

**IMPROVED ALGORIGHM**

After releasing the naïve algorithm we mentioned above. We find that both of the two approaching need to be improved in order to achieve better performance and faster reaction speed.

**Smart Monte Carlo Approach:**

Generally speaking, the method we used in Monte Carlo Gomoku Agent is Monte Carlo Tree Search. However, we adopted some other algorithms and did several important modifications to make the agent more reasonable and intelligent.

MONTE turn

Smart simulate

Best move

Win-time history

Update history win-time

**Pic 8: Monte basic process for one step(before improvement)**

At the very beginning, our intuitive assumptions are:

**<1>**MONTE is an AI agent who only knows the rules of the Gomoku. And one day he plays that game with an expert called MiniMax. He is too weak to play with MiniMax, because different from MONTE; MiniMax knows all the best strategy of Gomoku and has a lot of experience. However, MONTE has friends help him. One is an idiot as him called MONTA.(Here MONTA plays randomly)

**<2> (Basic: Monte plays thousands games randomly and records the result, then select the best moves)**

When MONTE is playing with MiniMax, MONTE and MONTA randomly try legal Gomoku moves till the game ends thousands of times, and learn from what they have tried. There are situations that the game ends: (1) Win. (2) Lose. (3) Tie. The two play thousands times of the Gomoku game. MONTE gets “experience” based on the history win-time of every move stored in his memory. Each move is associated with a win-time. In order to do this, we updated all the win-time values of all played moves in one simulated game when one game ends. If "win", the win-time of all tried moves in this simulated game increased by 1, which is a reward; if "lost", the win-time decreased by 2, which is a punishment; if "Tie", win-time does not change. After updating the win-time, MONTE can find the move associated with highest win-time and then play it.

**Pic 9: Monte’s detail for the smart simulating before improvement**

Thousands of times

For each possible moves

Simulate with MONTA to the end

Simulate with MONTA to the end

Simulate with MONTA to the end

……

**<3>(Improvement: Select 20 best previous moves as the initial moves and play random games, then select the best moves)**

MONTE finds that not all possible moves need to play; only part of moves can be the candidates of the best moves. He tries to select 20 best previous moves as the initial moves in playing random games. These 20 moves showed their good performance in previous simulations. That gives them qualification to become the "roots". But that does not mean, after the simulating of this round, these 20 moves can still has some higher win-time value. After simulating, the rank of best moves will change, and MONTE found the “best” move (the highest win-time move) and he will play it!

Thousands of times

Play that move

Select the best previous 20 moves

Simulate with MONTA to the end

Simulate with MONTA to the end

Simulate with MONTA to the end

……

**Pic 10: Monte’s detail for the smart simulating after <3>**

**<4> (Improvement: Add short cut)**

Assuming it is MONTE's turn, obviously one move will be played in this round is the move makes the MONTE wins immediately. Monte will choose that move directly without simulating as using our algorithm. Similarly, if one move will be played by the opponent makes MONTE lose immediately, Monte will choose that move directly without simulating as usingour algorithm. We call this process as “Shortcut". MONTE performs like this because he knows the basic rules of the game.

MONTE turn

One-step-win OR one-step-lose

Smart simulate

Yes shortcut

No

Best move

Win-time history

Update history win-time

**Pic 11: Monteprocess with short cut after <4>**

**<5> (Improvement: Add weak Minimax player as opponent for 1 move)**

MONTE now has a new friend, MM. MM is a weaker expert who knows some of the best strategy and some experience, but his time is limited. He can only train MONTE for several steps one time. So now every step playing with MiniMax, MM will train MONTE with one next move and then MONTE has to practice with MONTA as usual.

MONTE loves defending now, and previous problem like "do not know how to add one stone for an existing three stones in one line" is solved.

Thousands of times

MM plays one wise move

Play that move

Simulate with MONTA to the end

Simulate with MONTA to the end

Simulate with MONTA to the end

……

For the best 20 moves

**Pic 12: MM trained Monte’s detail for the smart simulating after <5>**

**<6> (Improvement: Repeat smart simulating and history win-time updating for several times and adding weight for current and history win-time)**

After several games with Minimax, MONTE finds that in some situation the previous 20 best moves have some misleading effect, so he thinks repeating the smart simulating and history win-time updating for several times before choosing the best move may be helpful for solving misleading. Also adding some weight in updating history win-time is another good choose (new history - win-time = history -win-time\* factor1 + smart – simulating - win-time\*factor2). For the history win-time, its weight is smaller than the smart simulating win-time's weight, which means the history data has sort of effect, but not too much. With this change, MONTE plays much better than before.

MONTE turn

one-step-win OR one-step-lose

Smart simulate

Yes shortcut

No

Best move

Win-time history

Update history win-time with weight

with

Repeat n times

**Pic 13: Monte process after <6>**

**Summary:**

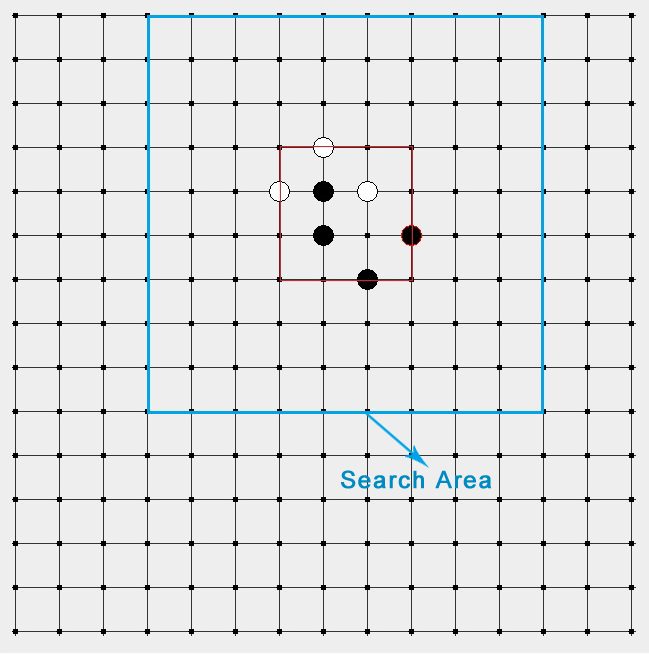
It is exiting that we find an OK solution for the Gomoku. It performs as well as Minimax in this case. However, Gomoku only has 15\*15 possible moves totally, suppose that if we have a very large board, which means the search space is unable to travel using a complete Minimax tree, what we can do is using our new approaching here. That is using the combination of **Monte Carlo** and **Genetic Algorithm**, and trained by **a smaller Minimax tree**. And then we can get an approximated best solution. The thought even can be apply to all cases making decision with a search tree.

Another thing need to mention is that in the proposal, we mentioned that we want to use DBN. However, the test cases show bad performance than the model we have now. The reason, in my view, is the limited cases we simulate. Comparing with the whole search space 3225, we only simulate at most 60,000 times, which cannot approximate the acceptable weight of every location. As a result, we keep the history win-time, and the win-time of the prior simulating will affect the entire future steps.

Also we try different ways of simulating, which will analyze later with our tests results.

**Fast Min-Max Approach:**

The tradition approaching of the Minimax is to search all the possible moves for the next step, and build a Minimax search tree of the depth considering the whole board. Here is the Gomoku, the thing has some difference. Due the rules of this game, the best next move is always near the area where there has already been played before. Taking that character into consideration, we limited our Minimax searching area to a rectangle which is three moves larger both in height and in width than the smallest closures of the existing moves.

**Pic 11: modified Minimax search area for certain step**

**SUMMARY**

After this modification, the speed of our fast Minimax is doubled what we have before. Also, he can wisely make the same decision as he did before. The comparing data also will be mention in the test result.

**TEST RESULT**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | opponent | Win rate W:L:T  (n Games) | Monte Average time per step | MiniMaxAverage time per step | Average step per Game |
| Simulating Game For Each Root Move:1000  Repeat Smart Simulating : 1  MiniMax Training Depth: 2 | Minimax(3) | 14:15:0(30)  Win:46%  Lose:50%  Tie:3.33% | 1.13s | 5.48s | 74 |
| Simulating Game For Each Root Move:5000  Repeat Smart Simulating : 1  MiniMax Training Depth: 2 | Minimax(3) | 18:10:2(30)  Win:60%  Lose:33.33%  Tie:6.67% | 1.58s | 5.21s | 75 |
| Simulating Game For Each Root Move:10000  Repeat Smart Simulating : 1  MiniMax Training Depth: 2 | Minimax(3) | 7:2:1(10)  Win:70%  Lose:20%  Tie:10% | 2.46s | 5.7s | 92 |
| Simulating Game For Each Root Move:1000  Repeat Smart Simulating : 3  MiniMax Training Depth: 2 | Minimax(3) | 16:11:3(30)  Win:53%  Lose:36.37%  Tie:10% | 3.43s | 4.67s | 82 |
| Simulating Game For Each Root Move:5000  Repeat Smart Simulating : 3  MiniMax Training Depth: 2 | Minimax(3) | 7:1:1(10)  Win:70%  Lose:10%  Tie:10% | 6.61s | 8.57s | 113 |
| Simulating Game For Each Root Move:10000  Repeat Smart Simulating : 3  MiniMax Training Depth: 2 | Minimax(3) | 21:6:3(30) | 7.52 | 5.73s |  |
| Simulating Game For Each Root move trained by Minimax(2) twice and simulates  Simulating Game For Each Root Move:20000  Repeat Smart Simulating : 1  MiniMax Training Depth: 2 | Minimax(3) | 1:29(30) | 5.11s | 5.68s | 15 |

**Table1: Comparison for different approaching of Monte**

Test Environment: CPU Corei5, Memory 6GB

Test Argument: Win reward 1, Lose punishment 2, Tie value 0, Smart simulating win-time weight 0.6, History win-time weight 0.4

**SUMMARY**

In the first three lines, the MONTE play better and better with the times of simulating increased. The more games MONTE trying means the larger chance per location can be played. And MONTE will gain better weight of every location, like more useful experience.

Comparing line 1-3 with line 1-4, the result shows that more repeat for smart simulating increasing the win rate of MONTE. That is because repeating smart simulating for several times filters the dummy best moves, which are good in the previous moves but not in this move, and the true best moves come out.

Interestingly, if we will try to let the MM trains he twice, the Monte did worse. Why?

Suppose our win-rule is five stone in row, and generally speaking three good moves can lead one player win immediately. If the MM trains MONTE two more moves, MONTE will have a pretty low probability to win (quite near Zero), which means he can hardly gain reward, as a result he cannot make a wise choice then. Also let us think like human beings. That training means the MONTE is a primary school student who is trained to use calculus. No surprise that he will perform worse.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Search area | Search tree node Count for 1st step | Avg search tree node count till 5st step/step | Avg search Tree node count till Game End/step |
| Unimproved-Minimax(3) | Whole board | 1597360 | 1013302 | 642326 |
| Improved-Minimax(3) | Limited area | 73870 | 135515 | 128551 |

**Table2: Comparison for different approaching of Minimax**

**SUMMARY**

Comparing the different approaching of Minimax, at the start of the game, the improved Minimax performs much better than the unimproved Minimax. This is because the improved Minimax 's search area is only part of the board. When the chess board contains more and more chesses, the improved Minimax 's serach area is expanded and its search tree nodes increase. This narrows the gap of the improved Minimax and unimproved Minimax. However, from the last column data, which represents the final stat of the game, the improved Minimax still is pretty better than the unimproved one.

**FUTURE**

In the process of our project, we have some thoughts for the further research. Because our limited time and back ground knowledge, we cannot finish it

* We find both history win-time and current simulating win-time has some effect on the current best move. Currently, we just use 0.4 as history weight and 0.6 as current weight in the Test. In the future, we want to use some other methods to calculate the new win-time list. One possible way is to map history and current win-time to [-1000, 1000](this normalizing need to study deeper to the Probability Model. We try to use equal probability model, but failed.).
* More works need to mathematically calculate the specific time of simulating led to the Maximum win rate.
* Also, here we use the reward 1 and punishment 2 in the win and lose cases, which is needed to be proved. Maybe other reward-punish strategy is better.

MONTE turn

one-step-win OR one-step-lose

Smart simulate

Yes shortcut

No

Best move

Win-time history

Other win-time

updating methods

Repeat n times

**Pic 14: future Monte’s process**

**CONCLUSION**

Our proposal of this project is trying to approach the Gomoku is different ways. And we achieve our original goal and find more interesting phenomenon. It is an interesting, basic, challenging, and meaningful project. Reasons are as follow:

**Firstly,** the Gomoku is a game of our interest. It has simple rules, so that it is valuable to apply the basic algorithm and easy to find out better performance among different approaches.

**Secondly,** we are attempting a new solution for the game, and comparing with the classic one, which is helpful for us study deeply on the Min-Max and Monte Carlo methods.

And we successfully provide an alternative method that can deal with the situation where the search space is too large to build the whole Minimax search tree. In that case, we can use a local Minimax searching tree combining with MonteCarlo simulating. Also, further search can be done as we mentioned.

**Last** but not least, as Gomoku is one of the Reversi. The new attempt on Monte Carlo might benefit to other broad game, like go.

**REFERENCE:**

1. [BC01] BrunoBouzy, Tristan Cazenave. Computer Go: an AI Oriented Survey. Artificial Intelligence, Vol. 132(1), pp. 39-103, October 2001.

2. [Brü93] Bernd Brügmann. Monte Carlo Go. Computer Go Magazine, 1993.

3. [SDS00] Nicol N. Schraudolph, Peter Dayan, Terrence J. Sejnowski. Learning To Evaluate Go Positions Via Temporal Difference Methods. IDSIA-05-00, 2000.

4.D. Erbach, Computers and Go, in The Go Player’s Almanac, ed. R. Bozulich (The IshiPress, 1992) 205–17

5.D. Fotland, private communication

6.N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, Equationsof state calculations by fast computing machines, J. Chem. Phys. 22 (1953) 1087-92