Gomoku Competition 2019

16307130214 Baichuan LIU
16307110288 Chuan ZHANG

Abstract—Gomoku is a simple zero-sum adversarial chess-playing game. Some classical methods like Minimax Search, Genetic algorithm and Monte Carlo Tree Search are proposed to solve this problem.
We designed several AI and figured out the most sufficient one which is developed on Minmax algorithm. Then We use some algorithm optimization methods including candidate selection, alpha - beta pruning,VCT and random threat space search to improve the performance.

Index Terms—Minimax, Gomoku, Threat space search, MCTS

1 Introduction

Gomoku is a well-known and classical board game which is extremely popular around the world. This game is played by two players, one with black pieces and the other with white pieces, within a 20×20 square mesh. The players place their own pieces by turn, and the first one who can achieve a 'five' will win the game. 'Five' means that at least five adjacent pieces of yours placed horizontally, vertically or diagonally within the mesh.

There have been several classical methods for AI to deal with this game. One of the most easy but powerful class of methods is greedy search algorithm. Minimax search, Genetic Algorithm and Monte Carlo Tree Search are widely used to solve this problem. Not only the Gomoku, many other cheese games can use these algorithms, for example, the Alpha Go uses Monte Carlo Tree Search together with reinforcement learning. Some neural network methods, such as the CNN applied in reinforcement learning, are also found to be useful.

In this report, for the limitation of time and device, we prefer to try greedy search algorithms than neural network methods. Specifically, we tried Minimax Search and Monte Carlo Tree Search together with some tools including $\alpha - \beta$ pruning and Threat Space Search. And we finally used the model developed based on Minimax Search.

In following section, we will review the basic Minmax search and Monte Carlo Tree Search algorithms and then introduce our evaluation policy as well as other strategies.

2 Minmax Search

2.1 Introduction

We can formulate the state space by scanning details of the board.

In Minmax algorithm, we can search all possible moves with a certain depth, and choose the best move we already know. The main idea is calculating the maxmin value. For min-nodes, which represents the opponent's lowest values representing moves my AI has to choose while for max-nodes, which represents the my highest values representing moves my AI can choose.

2.2 Candidate Selection

Candidates stand for those possible moves under a certain state. Theoretically we can find the whole state space for a Gomoku problem, which indicates that we can find out all the candidates in any states.

In practice, however, state space is so large that sometimes we have no ability to extend all of them. For instance, if we can choose any vacant position on board, we have to consider 400^4 board states. This will exceed time and memory limit absolutely.

Thus we have to reduce branches by selecting candidates in each layer. One of the most natural idea is to find those positions which are near to the existed chesspieces. So we do the process in the following way:

1) Step1: Find out all the positions who have a chesspiece on it, no matter black or white.
2) Step2: Add their neighbor spare positions (also can be regarded as the candidate actions) into a set.
3) Step3: Based on evaluation, we choose those moves who have largest scores as the candidates. In our AI, we choose top7 candidates regarded the time and memory limit.

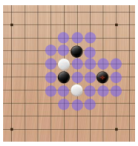

Fig. 1. Around the set pieces

Plus, we find that sometimes choose only the nearest positions as candidates is far from the optimal policy, for

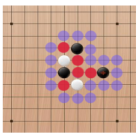

Fig. 2. Top 5 candidate

TABLE 1
Pattern recognition

Pattern	Our Score	Enemy's Score
11111	1000000	1000000
011110	20000	100000
011112	6100	7000
0101110	6000	7000
001110	1100	5500
010110	1000	5000
31	-100	-100
...	...	...

example, we can create a double four threat by place the piece on the middle position of a sequence "111000111", but this position is at least 2 blocks far from any existed pieces. To tackle with this problem, we enable the 2 blocks far positions to be regarded candidates. This will extend our search space quite a lot, so we only do this with a probability.

2.3 Evaluation

2.3.1 Point Evaluation

In order to choose candidates,we have to score every possible position. Here, pattern recognition is important prior knowledge.
Part of our patterns are shown in table1.

In point evaluation, we check four directions of certain point, score the line if it satisfies specific patterns. Every line will be evaluated by two directions(for example:from top to down and from down to top for column).I also flip the board to evaluate again which means I can prevent rival from getting high scores if I set my piece here.

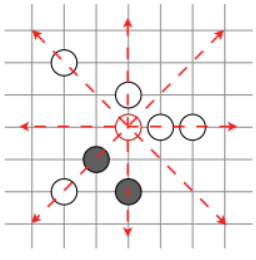

Fig. 3. Point evaluation

2.3.2 Board Evaluation

There is sight difference between point evaluation and board evaluation. In board evaluation, I just sum the score of all lines(all rows,all columns,all main-diagonals,all sub-diagonals). We also flip the whole board and sub the score rival can obtain.

2.4 Alpha-Beta Pruning

We apply classical alpha-beta pruning to reduce search time. Alpha-Beta pruning is a search algorithm that seeks to decrease the number of nodes. It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move.

```
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
```

Fig. 4. Alpha-Beta Pruning

2.5 Force Move and Threats Space Search

There are such cases in Gomoku, where one player keep attacking with force move and the other one has no other way but to do blocking - the forced move, such as straight-four, double three and double four (5). We implement force move methods to continuously pose direct threats to force rival to block. To prevent rival from doing force move, we block rival as early as possible.

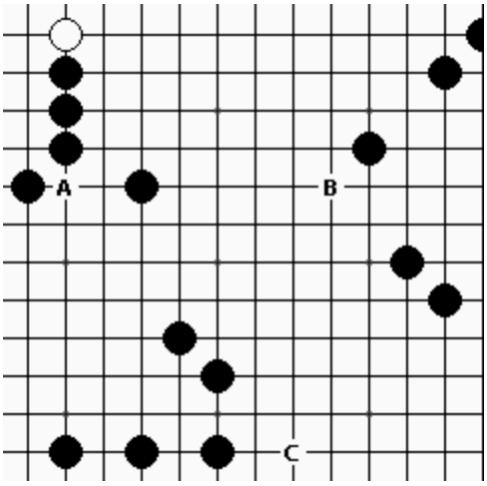

Fig. 5. Examples for threat sequences

Instead of the points chosen by candidate selection, its nodes are those who pose direct threat. As the expanding

nodes decrease significantly, this algorithm would not spend too much time.

To prevent rival from knowing and learning our policy, we add randomness to the policies. Execute TSS(threat space search) method in first layer to find only candidate which prevents rival from formulating threat with a certain probability. In this way, not only can we apply the threats space search method in a proper way which will not take a very long time, but we do add some randomness to our policy such that our policy can be variable even in the same situations.

3 Monte Carlo Tree Search

3.1 Introduction

Monte Carlo Tree Search (MCTS) is based on Monte Carlo method, which is a classic way for simulation. Its idea is to use simulations to evaluate the value of a node, so it requires simulating until the end and quite a lot samples.

Briefly, the process can be concluded into four main steps:

1) Selection: selecting the most urgent expandable node;
2) Expansion: expanding child nodes for the tree (finding candidates);
3) Simulation: simulating from the leaf to the end based on settled policy to evaluate the score;
4) Backpropagation: propagate the value to the original leaf node.

The whole process is a bit similar to what we have done in Mimmax Search, and the biggest difference between these two methods is that now we use simulations to replace the heuristic function (evaluation function) which is setted empirically before.

3.2 UCB

Here we use Upper Confidence Bounds (UCB) algorithm to balance the intention for exploration and exploitation, and this algorithm will help us to find the result faster. The formula is:

$$UCB = Q_j + \sqrt{\frac{2\ln(n)}{n_j}}$$

Where Q_j is the average reward from j_th simulation, n_j is the number of times that node is visited and n is the total number.

3.3 Algorithm

After defining the reward formula, we can show the whole MCTS peusdo code as figure 6.

The details are quite similar to the ones mentioned in Minmax Search. Besides, this is the brief skeleton of MCTS, in practice, we also don't need to do simulations every time, cause so much faster and sufficient way such as TSS also can be used before MCTS.

Algorithm 2 The UCT algorithm.

```
function UctSearch(s_0)
    create root node v_0 with state s_0
    while within computational budget do
        v_l ← TreePolicy(v_0)
        Δ ← DefaultPolicy(s(v_l))
        Backup(v_l, Δ)
    return a(BestChild(v_0, 0))

function TreePolicy(v)
    while v is nonterminal do
        if v not fully expanded then
            return Expand(v)
        else
            v ← BestChild(v, Cp)
    return v

function Expand(v)
    choose a ∈ untried actions from A(s(v))
    add a new child v' to v
        with s(v') = f(s(v), a)
        and a(v') = a
    return v'

function BestChild(v, c)
    return argmax_{v' ∈ children of v} Q(v')/N(v') + c sqrt(2 ln N(v) / N(v'))

function DefaultPolicy(s)
    while s is non-terminal do
        choose a ∈ A(s) uniformly at random
        s ← f(s, a)
    return reward for state s

function Backup(v, Δ)
    while v is not null do
        N(v) ← N(v) + 1
        Q(v) ← Q(v) + Δ(v, p)
        v ← parent of v
```

Fig. 6. MCTS algorithm

3.4 Comparison with Minmax

Unfortunately, MCTS didn't give a good performance due to the time limit. Cause the algorithm require to simulate to the very end to get rewards, every simulation need a huge amount of time, especially in the very beginning. WIthin 15 seconds, our AI can only complete around 10 20 times of simulations, so the performance is not that good.

Comparing to MCTS, Minmax Search only need to visit several layers, so it can be finished much faster. Considering the limitation of time, we give up MCTS and put most of our attentions on Minmax Search method.

4 Conclusion

These two methods have been introduced in our class, and Gomoku is a very good project to help us understand more about them. Moreover, we find that thouth simple, these two algorithms perform well on the chess problem, and we even cannot beat our AI! That's amazing and we have achieved a strong sense of achievement.

In a nutshell, Minmax Search is a very powerful method on Gomoku, especially when there are acute time limits, and we have applied several useful strategies to help improved the performance of our AI.

5 Reference

[1]Zhentao Tang, Dongbin Zhao, Kun Shao, Le Lv,ADP with MCTS Algorithm for Gomoku. 2016 IEEE

[2]Dongbin Zhao, Zhen Zhang, Yujie Dai, Self-teaching adaptive dynamic programming for Gomoku. Neurocomputing 78 (2012) 2329

[3]Chih-Hung Chen, Shun-Shii Lin, and Yen-Chi Chen, An Algorithmic Design and Implementation of Outer-Open Gomoku. The 2nd International Conference on Computer and Communication Systems

[4]Louis Victor Allis and Hj Van Den Herik, Go-moku and threat-space search(1993),