Gomoku

Zhang Jiwen 16307110435 WanYin 16307110431

I. Introduction

Gomoku, sometimes known as "five wins" is a game played on a 15x15 board by two players. The goal is simple: to be the first player to have a line of five stones.

To have an efficient Gomoku AI, we first explored various algorithms for Gomoku, such as Monte Carlo Tree Search, Genetic Algorithm, Reinforcement Learning, etc.

After fully exploring the advantages and disadvantages of the algorithm, we found that the secret of the victory of Gomoku is to have some expert knowledge and use it to evaluate the current board state. So we turned our attention to the scoring system and search algorithm combined with RL.

The first version of our AI can defeat Mushroom easily. After that, we spent about two days constantly enhancing it, mainly in two areas. Firstly, speed up our AI by reducing evaluating costs. Secondly, use a "default policy" to deal with "must to be blocked" or "must to be taken" cases.

After all, our AI can best beat EULRING16 with a winning percentage of 20.

II. Algorithm Exploration

A. *Genetic Algorithm*

a. Architecture

The following flow chart generalizes Genetic Algorithm in Gomoku.

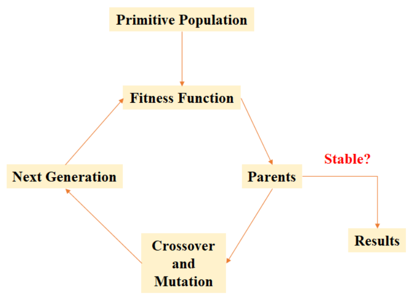


Fig. 1. Genetic Algorithm

When it is turn for our agent to make a move, it first generates a primitive population of DNAs (like a gene pool), where potential solutions lie. Then, after being evaluated by the fitness function, several best DNAs, called parents, are selected to evolve the population, i.e. by crossover and mutation.

The next generation will again be evaluated and the evolution process continues until the population stabilizes. At this time, our best move is found.

b. Coding Scheme

Our Gomoku board is 20 × 20. In our GA framework, we set our DNA length as 7, i.e. we consider seven consecutive steps from the current board to choose the best next move.

For example, after the human player makes a move at (7,5), our GA generates the best individual, i.e. a sequence of the subsequent seven steps like {(9,4), (7,6), (6,4), (7,7), (8,4), (7,4), (7,8)}, called a DNA . Then a signal to move at (9,4) will be sent to our agent. Subsequently, the human player makes another move and the whole GA framework will be executed again to find the best next move.

c. Primitive Population

Given the current board, let (x_L,x_R) and (y_B,y_T) be the range along the horizontal and vertical dimensions of occupied positions on the board.

Since our DNA length is set as 7, we restrict the next seven DNA genes to be placed in the expanded boundary of (x_L-1,x_R+1) and (y_B-1,y_T+1). The positions of the genes must be vacant on the board in order to guarantee the validness of DNAs.

d. Fitness Function

Our fitness function is designed based on some prior knowledge of Gomoku. When some patterns appear on the board, it will be easier to judge whether we will soon taste the victory or fall into danger. Each pattern corresponds to a fitness reward. The reward is positive if our victory approaches, while it is negative if there will be a danger. Then, our agent can choose to attack or defend after considering the corresponding fitness rewards of all potential individuals. Here are some useful patterns for Gomoku.

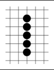
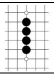
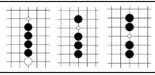
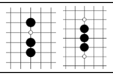
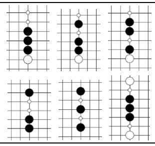
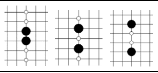
ID	Pattern	Fitness Reward
1		100000
2		10000
3		100
4		200
5		50
6		5

Table 1. Some patterns for Gomoku and fitness values

e. Crossover and Mutation

Crossover and mutation can add some randomness to the selected DNAs. In our GA framework, both crossover point and mutation point are selected randomly. Similar to natural selection, better DNAs can be generated after good ones crossover with each other. Then some genes in the DNAs will be mutated and replaced by another genes in the gene pool.

B. *Monte Carlo Tree Search*

Core idea of MCTS is taking random simulations in the decision space and building a search tree according to the results. It consists of four main stages: Selection, Expansion, Simulation, and Backpropagation

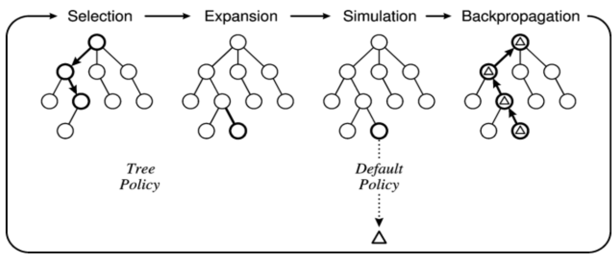


Fig. 2. The basic process of MCTS

- **Selection:** Begins from the root node, and recursively applies the child selection policy (also known as Tree Policy) to descend through the tree until it reaches the most urgent expandable node.
- **Expansion:** Add one or more child nodes to expand the tree.
- **Simulation:** Run a simulation from the leaf node based on the settled policy (or called Default Policy) to produce an outcome
- **Backpropagation:** Back propagate the simulation result through the selected nodes to update their state values.

Monte Carlo Tree Search Algorithm requires too much simulations. It needs some complex domain knowledge additionally, and is highly likely to spend lots of time in useless simulation.

Therefore, results given by MCTS were far from satisfactory.

III. Expert Knowledge

We derive the expert knowledge from HERE(https://zjh776.iteye.com/blog/1979748).

Using expert knowledge, we design a score dictionary for each "five-row" pattern:

Pattern	Score
Win	100000
H4	10000
C4	100
H3	200
M3	50
H2	5
M2	3
S4	-5
S3	-5
S2	-5

Table 2. Patterns-score Dictionary

Every time we want to do a board evaluation, we first scan the board to get all above 10 patterns. Then based on how many times a pattern appears, calculate the score of the board.

Note that the pattern recognition is based on current player is who. So to get a useful score, we need to scan the board twice, each time getting a score for one certain player.

IV. Min-Max Search

The relatively competitive and stable algorithm we found is Mini-Max Tree Search, which is also known as adversarial search. The basic idea is covered in our course. Here we give some details to make this algorithm work:

- **Expansion/Search**

Given a board, all places that has been occupied are seed nodes.

- Neighbour Nodes: Nodes that are *one-step* away from seed nodes.
- Adjacent Nodes: Neighbour nodes that has not been occupied.

For example, 8 places surrounding the center red node are neighbour nodes.

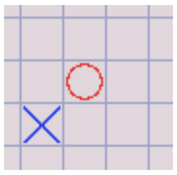


Fig. 3. Sample Board

Removing the blue node, the left are adjacent nodes.

Adjacent nodes as potential actions our Gomoku AI may take. We use these nodes to do the following evaluation.

- **Leaf Node Evaluation**

a) Heuristic

The evaluation at a leaf node is defined as follows:

$$score = score_{AI}(p) - score_{opp}(p)$$

where *opp* denotes the opponent of our AI, and p denotes the position of node.

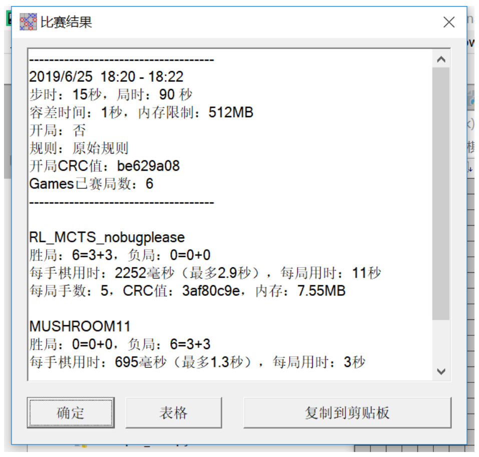
b) Reinforcement Learning

The score of adjacent nodes (also known as root nodes) should not only decided by its current board evaluation. It is also influence by its opponents' action in the next round. Inspired by this, we create a sub-tree of each root node. Leaf nodes of a sub-tree are adjacent nodes of current board.

Score of a certain root node will be updated when doing depth-limited searching. Q-Learning algorithm is applied:

$$\Delta score_{root} = \alpha * \gamma^{d} * diff$$

where α is the learning rate, γ is the discount factor, $diff$ is the difference of board score before and after a leaf action is taken. Using this AI we can defeat Mushroom easily without specifying a start.



```
------------------------------------
2019/6/25  18:20 - 18:22
步时：15秒，局时：90 秒
容差时间：1秒，内存限制：512MB
开局：否
规则：原始规则
开局CRC值：be629a08
Games已赛局数：6
------------------------------------

RL_MCTS_nobugplease
胜局：6=3+3，负局：0=0+0
每手棋用时：2252毫秒（最多2.9秒），每局用时：11秒
每局手数：5，CRC值：3af80c9e，内存：7.55MB

MUSHROOM11
胜局：0=0+0，负局：6=3+3
每手棋用时：695毫秒（最多1.3秒），每局用时：3秒
```

Fig. 4. Outcome of AI Version 1

V. Algorithm Acceleration

A. *Default Policy*

This accelerating function is inspired by the fitness function in GA or the score function in MCTS. Thanks to some prior knowledge of Gomoku, the function enables our agent to make a quick move once some patterns on the board appear after scanning the board by row, column and two diagonals.

First, we will check our AI's state on the board. If pattern 1, 2 or 3 in Table 1 appears on the current board, it means there is only one step away from success and the next move is definite.

The next is to check our opponent's state on the board. If pattern 1, 2, 3, or 4 appears on the current board, it means our opponent is approaching success. Hence, we have no alternative but to make a defending move.

Additionally, if both our opponent and our AI are not approaching win or loss, i.e. there is still a long way to end this game, our tree search will start to work in order to find the best next move.

B. *Cross Evaluation*

Board is evaluated to get a score when we have root nodes and would like to create sub-tree so as to update score of root nodes.

Suppose the board size is h and the scale of adjacent nodes is n, time complexity of the *get_action()* function is $O(n^2 * h^2)$. However, the whole board is not worth evaluating because a move only affects two crosses centered on it.

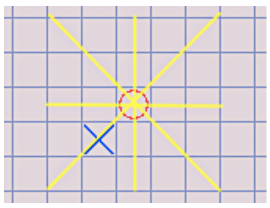
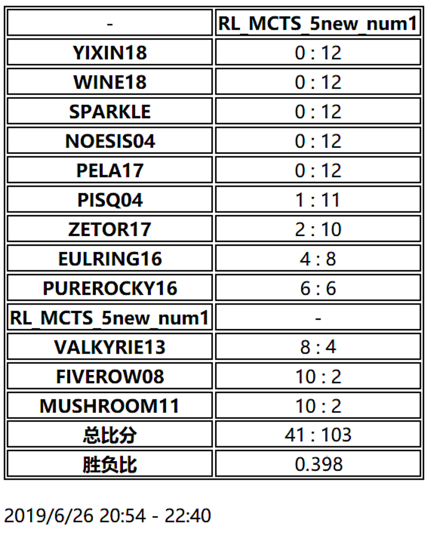


Fig. 5. Cross Evaluation on Board

Therefore, we only need to scan such two crosses to get the new score. This operation can greatly reduce algorithm complexity from $O(h^2)$ to $O(h)$.

VI. Tournament Outcome

The final outcome of our AI agent is:



-	RL_MCTS_5new_num1
YIXIN18	0 : 12
WINE18	0 : 12
SPARKLE	0 : 12
NOESIS04	0 : 12
PELA17	0 : 12
PISQ04	1 : 11
ZETOR17	2 : 10
EULRING16	4 : 8
PUREROCKY16	6 : 6
RL_MCTS_5new_num1	-
VALKYRIE13	8 : 4
FIVEROW08	10 : 2
MUSHROOM11	10 : 2
总比分	41 : 103
胜负比	0.398

2019/6/26 20:54 - 22:40

Table. 3. Outcomes

References

[1] Junru Wang, Lan Huangb, Evolving Gomoku Solver by Genetic Algorithm, IEEE WARTIA,2014

[2] Tang, Zhentao Zhao, Dongbin Shao, Kun Lv, Le. (2016). ADP with MCTS algorithm for Gomoku. 1-7. 10.1109/SSCI.2016.7849371.

[3] Louis Victor Allis,Searching for Solutions in Games and Articial Intelligence,Version 8.0 of July 1, 1994