MCTS-ADP Exponential Heuristic Algorithm for Solving Gomoku*

林阳昊 16307130122 曹旭 16307110230

Abstract—**We combine a neural network, which is trained by Adaptive Dynamic Programming (ADP) and a new sort of heuristic function with Monte Carlo Tree Search (MCTS) algorithm for Gomoku. MCTS algorithm is based on Monte Carlo simulation method, which uses selections and simulations to generate a game search tree. We apply pruning strategy and heuristic to select available point of the state and use ADP to evaluate winning rates at the end of simulation. Experiment result shows that this method can effectively eliminate the neural network evaluation function's "short-sighted" defect. The method can be applied to design new Gomoku AI.**

Keywords—adaptive dynamic programming; Monte Carlo tree search; Gomoku; Exponential Heuristic

I. Introduction

Gomoku is a traditional strategical board game designed for two players playing on the 15x15 go-board or other similar board. The players are alternatively putting their own signs (cross and circle or black and white chessman) on the board until one of them first manages a continuous line of five or more stones in the vertical, horizontal or diagonal direction and due to becomes the winner. This is the freestyle rules of gomoku.

To solve the freestyle gomoku, we design a new exponential heuristic function and combine it with two classic algorithms — the Monte Carlo Tree Search and Adaptive Dynamic Programming. The Monte-Carlo Tree Search (MCTS) is one of the most famous search algorithms for strategical board game. Most of times, it is better than other traditional tree search algorithms such as Minimax Tree Search with Alpha-Beta Pruning. However, MCTS still have some drawbacks which restrict its search ability. One of the most significant problems is the time and space complexity growing exponentially with simulation search depth. If MCTS's simulation depth choice and random sampling strategy are improper, it will cause a high time and space complexity. To handle this problem, we propose Adaptive Dynamic Programming (ADP) trained by shallow forward neural network and a Progressive Bias-UCB based on heuristic strategy and pruning strategy to improve the MCTS simulation efficiency.

Next section provides a brief description of the MCTS and ADP research related work, then section 3 briefly introduces the MCTS algorithm based on UCB. Section 4 presents our MCTS-ADP Exponential Heuristic Gomoku solver. Section 5 presents and discusses the experimental and competition results among our AI and other Gomoku AI solvers. In the final section, we summarize the whole paper and offered some possible improvement direction for future research.

II. RELATED WORKS

Monte Carlo Tree Search (MCTS) is a method for finding optimal decisions in a given domain by taking monte carlo simulations in the decision space and building a search tree according to the results. Paper [1] introduced some pruning and improvement of UCT method of MCTS.

Reinforcement learning is a novel machine learning method which concerns how agent ought to take actions in an unknown environment so as to maximize some notions of cumulative reward. The most competitive advantage of reinforcement learning is that it does not need knowledge about the Markov decision process (MDP) and can target the large MDPs when exact methods become fairly complex, such as Poker and Go. Dongbin Zhao [2] applied Adaptive Dynamic Programing (ADP) based on neutral network to learn to play Gomoku. They advised to train gomoku AI model by playing against itself and it has approached the candidate Level of a commercial Gomoku program called 5-star Gomoku. Then, Zhentao Tang [3] combined Monte Carlo Tree Search with Adaptive Dynamic Programming algorithm. They use ADP to train a shallow neural network and calculate the weighted sum of ADP and its corresponding winning probability of MCTS.

Inspired by Zhentao Tang's research, we apply Monte Carlo Tree Search with ADP into Gomoku, as well as improving its structure and strategy. Accordingly, we actually obtain a strong and effective Gomoku-AI.

III. MONTE CARLO TREE SEARCH WITH UCB

Monte Carlo tree search (MCTS) is a tree search algorithm based on tree structure, which can be used to tradeoff the exploration and utilization, and it can explore huger space than any other tree search algorithm. MCTS is best-first search by applying a large number of random simulation. Therefore, it obtains high accuracy with random sampling and decreases time cost that spends for calculating evaluation values.

The 4 important processes of MCTS is shown in Fig. 1. They are Selection, Expansion, Simulation, and Backpropagation. The first step is Selection. The aim of selection is to choose one of the best nodes worth exploring in the tree. In Gomoku, the general strategy is to select the 2-

grid adjacent child nodes that are not explored in the board. We can use Upper Confidential Tree (UCT) method as our selection policy, which makes proper selection strategy using Upper Confidential Bound (UCB) that can tradeoff exploration and exploitation. (1) is the formula of UCB. In this function, N is total number of nodes, and n_i is number of child nodes belong to node i. v_i is the value that the total winning rate of node that has all results of child nodes. C is a constant value.

$$\text{UCB} = v_i + C\sqrt{\frac{\ln(N)}{n_i}} \tag{1}$$

The function can help AI select the child node which has the largest UCB value. Then at Expansion step, it is to put the chessman in the previously selected child node and create new child nodes. The general strategy is to randomly select an operation. The third step is Simulation, which start simulating from the new expended child node and getting result value that includes Win(+1) or Lose(+0) information in the end of the simulation. It is worth mentioning that the depth of the simulation tree can be unacceptably large, therefore we stop the simulation process once the depth exceeds the MAX_DEPTH value. To receive the result value of a pre-terminated simulation process, we utilize an evaluation function trained by ADP, which will be explained in the next section. The fourth step is Back-propagation, which is to back propagate the score of the node from the previous expansion to all the previous parent nodes, and update the quality value and visit times of these nodes to facilitate the calculation of the UCB value.

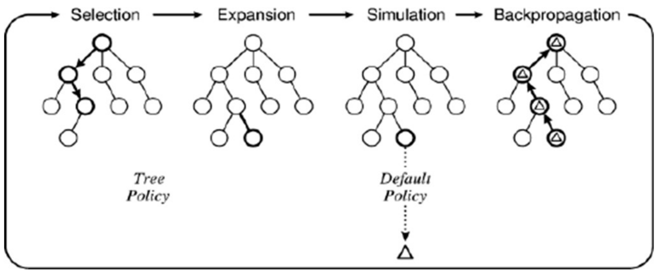


Fig. 1. MCTS process [1]

The other widely used MCTS algorithm is UCT [1], which is based on Upper Confidence Bounds (UCB). UCB is a well-known algorithm that can solve the multi-armed bandit problem. UCB is able to balance the conflict between exploration and exploitation and find out the final result earlier. It seems that UCB can solve the trade-off problem that we discussed above. Its simplest form of UCB is showing in formula (2). It just set the confidence c to $\sqrt{2}$ from formula (1).

$$\text{UCB} = \bar{X}_j + \sqrt{\frac{2\ln(n)}{n_j}} \tag{2}$$

where $\bar{X}_j$ is the average reward from j-th simulation, n_j is the number of times that node j is visited, and n is the overall number of plays so far. The reward $\bar{X}_j$ encourages the exploitation of higher reward selection, but the right-hand term $\sqrt{\frac{2\ln(n)}{n_j}}$ encourages the exploration of less visited choices.

UCT is originated from HMCTS, but it is more effective and complexed than HMCTS. UCB can help to find out the balanced leaf nodes earlier than simple HMCTS algorithm, thus, UCT can save more time than the original version.

IV. MCTS-ADP EXPONENTIAL HEURISTIC ALGORITHM

A. Selecting heuritic function

To reduce most of computing cost in this AI, it is significant to design an efficient heuristic algorithm which is simple and has practical form. The heuristic function needs to be able to evaluate the rewards of the next drop, including the offense rewards and the defense rewards. Since Gomoku is not a complicated game, it is not difficult to build such a heuristic function. Jun Hwan Kang and Hang Joon Kim [4] proposed to use formula (3) as the Gomoku heuristic function.

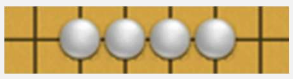
$$H_i = \sum_l \left\{ \left(L_{open}\right)^2 + \left(\frac{L_{hclose}}{2}\right)^2 \right\} \tag{3}$$

Variable L_{open} is length of line that has no opponent's chessman at two terminals, while variable L_{hclose} is length of line that has only one opponent's chessman at its terminals. However, this heuristic function has a bit of problem. First, it ignores the potential for discontinuous offense patterns, such as the one known as 'TIAOSAN'. Second, its valuation of offense is unreasonable. In freestyle rules of Gomoku, 'HUOSAN' ($L_{open} = 3$) and 'MIANSI' ($L_{hclose} = 4$) should have similar offense effect, but they differ too much in this heuristic (9 and 4).
To solve this problem, we design a new sort of heuristic function: exponential heuristic function. It can consider more states and situations than Jun Hwan Kang's simple heuristic.

$$H_i = \sum_l \{10^{L_{open}} * factor^j + 10^{L_{hclos}\ -1} * factor^k\} \tag{4}$$

Variable L_{open} is length of line that has no opponent's chessman at two terminals and some similar situation with a descend factor, while variable L_{hclose} is length of line that has only one opponent's chessman at its terminals and some similar situation with a descend factor. The factor is a constant. We set it as 0.90.

Use formula we obtain 29 patterns' heuristic values. The table below shows part of it. The full heuristic function and pattern is in our codes.

ID	Pattern	Value
1		10000

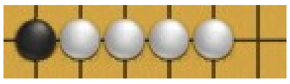
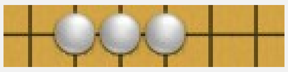
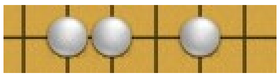
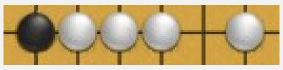
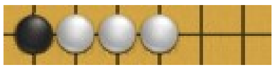
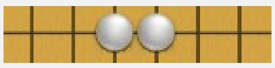
2		**1000**
3		**1000**
4		**1000 ∗ factor**
5		**1000 ∗ factor**
6		**100**
7		**100**

TABLE I. PART OF PATTERN REWARDS

B. *Selecting adjacent function*

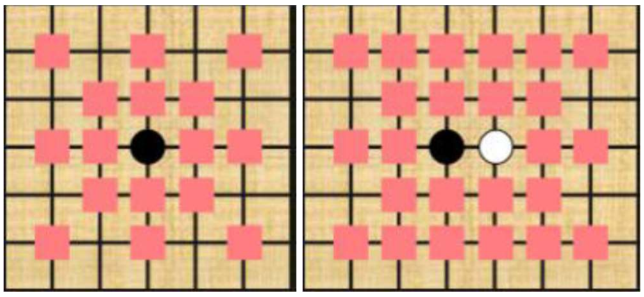


Fig. 2. Example of the adjacent point [7]

Because of less restriction in free-style Gomoku, the number of legal moves is pretty large. According to our designed heuristic function, the available point should be the 2-adjacent as the figure 2 shows. Supposed the available is 1-adjacent, it will miss some significant point such as the left chessman in pattern ID 4 in table 1.

C. *ADP*

To evaluate board situations, we utilize a Multi-Layer Perceptron (MLP) of 3 layers (Figure 3). The input features of the MLP includes the number of the 29 patterns mentioned above and a one hot vector indicating who is to move next. The number of patterns are encoded in a specific way, where a number is represented by a vector of size 5 (Table2). The activation function of the MLP is the sigmoid function:

$$g(x) = \frac{1}{1 + e^{-x}}$$

Which normalizes the output value to the range $(0, 1)$. This allows us to interpret the output of the MLP as the winning probability of a player.

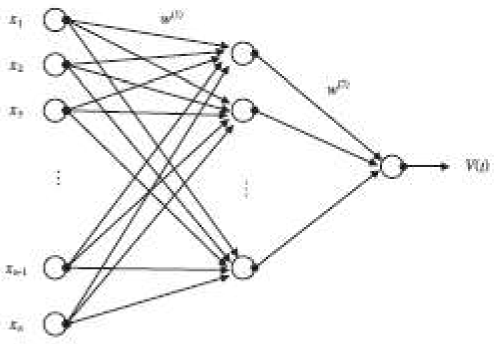


Fig. 3. The neurtal network of ADP [3]

Value of n	Input 1	Input 2	Input 3	Input 4	Input 5
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
>4	1	1	1	1	$\frac{n-4}{2}$

TABLE II. ADP INPUTS

We train the MLP in an Adaptive Dynamic Programming (ADP) process. The reward is set to 0 during the game. When a game ends, if the player wins, the reward is 1.Else if the opponent wins, the reward is 0. The prediction error is defined as

$$e(t) = \alpha[r(t+1) + \gamma V(t+1) - V(t)]$$

Where α is the learning rate and γ is the discount factor (set to 1). V is the output of the MLP.

V. EXPERIMENTS AND ANALYSIS

Agent	mushroom	PureRocky	Valkyrie	Pisq7
ADP	10:0	9:1	7:3	3:7
UCT	5:5	2:8	1:9	0:10
ADP-UCT	10:0	10:0	9:1	2:8

TABLE III. COMPARISON AGAINST OTHER AGENTS

We let our agents play against other Gomoku agents, including mushroom, PureRocky, Valkyrie, pisq7. Table 3 shows that ADP-UCT has better performance than both ADP and UCT. If we continue add heuristic function and pruning strategy, the efficient will increase to 9:3 of AI Pisq7.

The result implies that UCT is an effective support for ADP. ADP alone can make decent moves in a very short time (at around 100 milliseconds), while UCT improves the decisions made by ADP with the cost of time.

VI. CONCLUSION AND OUTLOOK

Overall, we refer a method by employing ADP combined with MCTS algorithm to solve Gomoku in this paper. From the experiment, ADP and exponential heuristic function with MCTS is able to win most of weak AI in gomocup. However, it still has a certain gap with some stronger AI such as YiXin and Wine. We will try to employ some tree shape control method to decrease high branch factor and apply CNN or LSTM to train a more powerful ADP.

REFERENCES

[1] Browne C B , Powley E , Whitehouse D , et al. A Survey of Monte Carlo Tree Search Methods[J]. IEEE Transactions on Computational Intelligence and AI in Games, 2012, 4(1):1-43.

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

[3] Zhao D, Zhang Z, Dai Y. Self-teaching adaptive dynamic programming for Gomoku[J]. Neurocomputing, 2012, 78(1):23-29.

[4] Kang, J.H.. (2016). Effective Monte-Carlo tree search strategies for Gomoku AI. 9. 4833-4841.

[5] Tan K L , Tan C H , Tan K C , et al. Adaptive game AI for Gomoku[C]. 2009 4th International Conference on Autonomous Robots and Agents. IEEE, 2009.

[6] Cowling P I , Powley E J , Whitehouse D . Information Set Monte Carlo Tree Search[J]. IEEE Transactions on Computational Intelligence & Ai in Games, 2012, 4(2):120-143.

[7] Chen C H , Lin S S , Chen Y C . An algorithmic design and implementation of outer-open gomoku[C]// 2017 2nd International Conference on Computer and Communication Systems (ICCCS). IEEE, 2017.