Gomuku Final Report
AB-Pruning & RL

Shao Yanjun 19307110036
Shao Yi 19307130113

2021-12-19

1 Abstract

The Gomoku agents we implemented are based on algorithms like Alpha-Beta Pruning (ABP), Monte Carlo Tree Search (MCTS) and Adaptive Dynamic Programming (ADP). Furthermore, we tried some advanced techniques like Upper Confidence bound applied to Trees (UCT) and Critic Network.

2 Feature Patterns

This is our proud design, we only focus on feature patterns below instead of global board information, and we also divide these patterns into the following categories: Winning Identity, Winning Threats, Forcing Threats and Potential Threats. Without loss of generaliry, we consider situations for the black.

2.1 Winning Identity & Winning Threats

Winning Identity is five-in-a-row, which terminates the game. Winning Threats include open fours and special cases, they are patterns that create two choices for the black to win, while the white only has one turn to defend.

Figure 1: Winning Identity & Winning Threats

2.2 Forcing Threats

Forcing Threats includes simple fours, open threes and simple threes, which force the white to respond with defense actions, otherwise the black may develop Winning Identity or Winning Threats next turn.

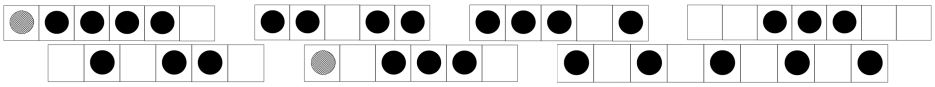

Figure 2: Forcing Threats

2.3 Potential Threats

Potential Threats includes broken threes and simple twos. They might not be decisive for the current situation, but they are non-negligible and can be runner-ups for a future winning pattern.

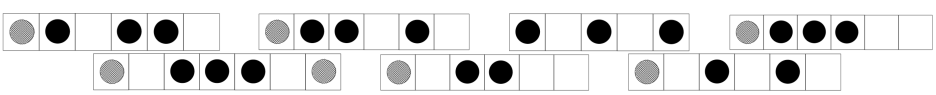

Figure 3: Potential Threats

3 Alpha-Beta Pruning Agent

For the final commit, the previously naive short-sighted alpha-beta pruning agent, namely pydan1, was refined and modified in order to compete for better rankings.
Generally speaking, the agent exploits a minimax search paradigm with delicate prunings, which might include alpha-beta pruning, threat-based pruning and heuristic pruning. Also, a special hashing technique named Zobrist is used to help cache the board.

3.1 Minimax

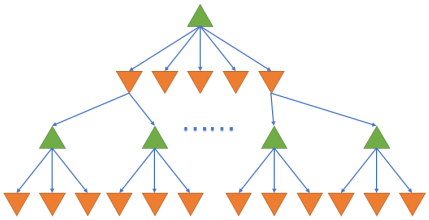

Figure 4: minimax tree search

Minimax search is broadly proved useful with the development of game theory. As is shown above, the root of the tree search is a max-node, which selects over the highest possible value returned from its min-node children. Each min-node then selects the minimum value within their reach. The bottom of the minimax tree consists of leaf-nodes, where search comes to an end. Each leaf-node returns an evaluation using the patterns and their corresponding value detailed in code.

3.2 Evaluation

Basically, the evaluation of a board state is an wighted sum of the patterns of the entire board. Mathematically, it should be formalized as follows,

$$SCORE = \gamma_1 \sum_{(p,v)\in P_1} |p| \cdot v + \gamma_2 \sum_{(p,v)\in P_2} |p| \cdot v$$

In this, variable p denotes the patterns generated by regular expressions and v denotes the corresponding value. $|\cdot|$ represents the number of patterns.
Both γ_1 and γ_2 are discount factors controls the balance of the evaluation. For instance, if the $SCORE$ is evaluated when P_1 plays the offensive role, we should have $0 < \gamma_2 < 1$ in that all the threats of P_2 have some tendency to be defended. Meanwhile, all the patterns of P_1 become more powerful, as P_1 can exploit this move to expand its advantages.

3.3 Exploration

It is really hard to do minimax search without the help of an efficient exploration function. To pass through the exploration node, the agent will search for candidates of blank space which falls within 2 reaches of existing pieces.

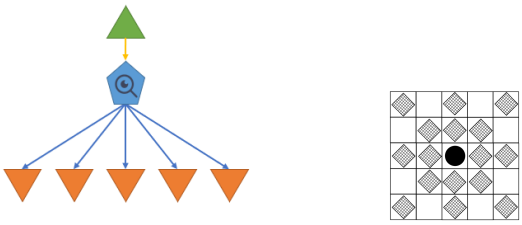

Figure 5: exploration

An improvement, at the same time, is that we explore the consistency of board game, which is, a single move on the Gomoku board only has impact on the horizontal, vertical, diagonal and anti-diagonal neighbours. Patterns can be dealt with locally at a low cost using regular expressions and we no longer need the poorly-implemented deepcopy of nested arrays in Python. Also, the specialization of an exploration process provide some insights for us to do threat-based prunings.

3.4 Alpha-Beta Pruning

With this special technique introduced in our first report, we can do prunings on a huge amount of nodes within the minimax tree. The pruned-nodes are guaranteed not necessary to be explored.

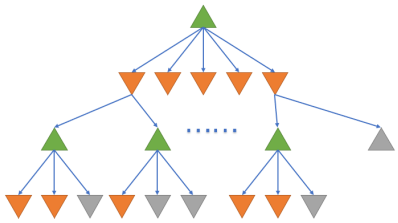

Figure 6: alpha-beta pruning

3.5 threat-based Pruning

Briefly speaking, four types of threat-based pruning techneque will induce another huge amount of guaranteed local prunings.

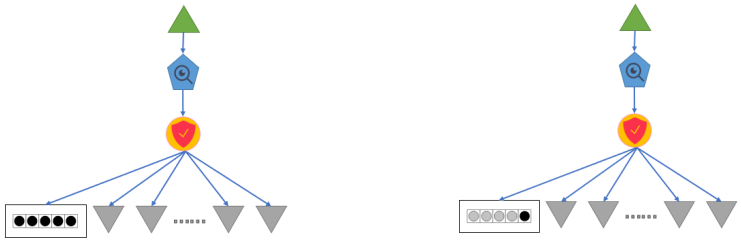

Figure 7: winning prune

The left part of the prune is instantaneous, as five in a row indicates complete victory. And the right part of the prune is compulsory, as P_1's failure to block the pattern of four directly leads to the loss of the game.

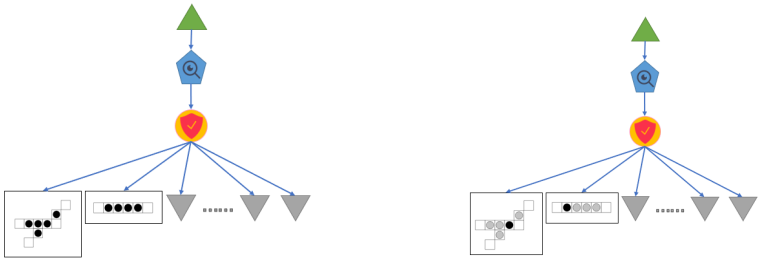

Figure 8: potential prune

These two prunings are not explicitly compulsory. However, if either P_1 or P_2 has an opportunity to extend a double three or an open four, it is already powerful enough to terminate the game. Local blockings or offending are, therefore, compulsory.

3.6 Heuristic pruning

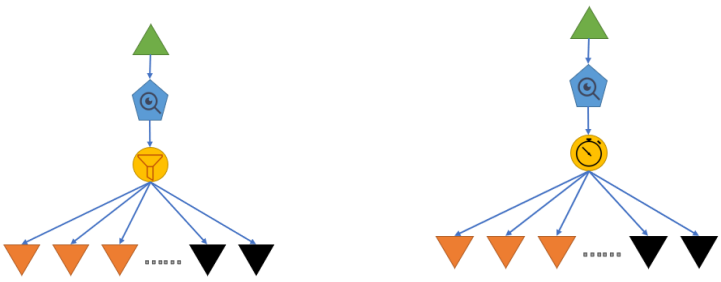

Figure 9: heuristic prune

These prunings are rather heuristic and resigned. To be more specific, as the board is updated at the stage of exploration, the evaluation is accessible. We can then sort by the local evaluations of the board and heuristically select the top K candidates. Also, if time is ellapsed, we can still return a quasi-optimal solution.

3.7 Hashing skills

A popular utility of the hashing skill, Zobrist, can capture uniquely the state of every board with a sufficiently large integer. We can exploit the efficiency of Python dictionary and Zobrist skills to cache the patterns and values of each board state. The searching cost will be dramatically reduced if a path has been explored before.

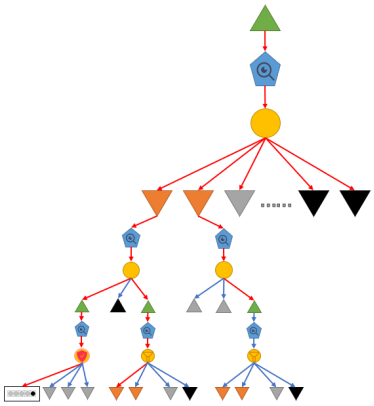

Figure 10: cached search

4 Reinforcement Learning Agent

Approximate Dynamic Programming (ADP) is a powerful technique to solve large scale discrete time multistage stochastic control processes, i.e., complex Markov Decision Processes (MDPs). These processes consists of a state space $\mathcal{S}$, and at each time step t, the system is in a particular state $S_t \in \mathcal{S}$ from which we can take a decision x_t from the feasible set $\mathcal{X}_t$.
The key idea of Adaptive Dynamic Programming is training Gomoku by temporal difference learning, as a result, the winning rate of a certain chess board state can be described in continuous form, approximated by nonlinear function, such as neural network in our works.

4.1 Adaptive Dynamic Programming

The ADP training structure is illustrated in the following figure. To solve the problem that the network converges very slowly and randomly, we try to obtain candidate action moves by ADP. Every one of candidate moves obtained from ADP is selected by basic pruning.

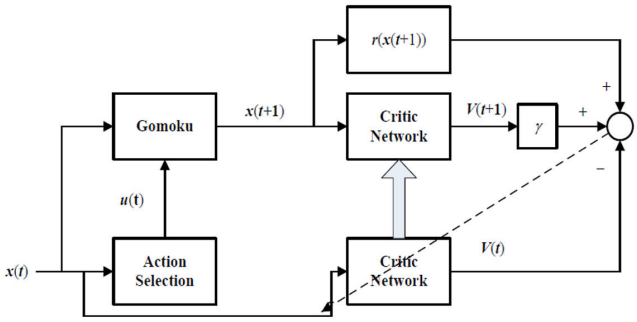

Figure 11: Adaptive Dynamic Programming

4.2 Critic Network

Our critic network is a multilayer perceptron with one input layer, one hidden layer and one output layer, and we simply use fully connected neural network between each adjacent layers, the structure is shown below.

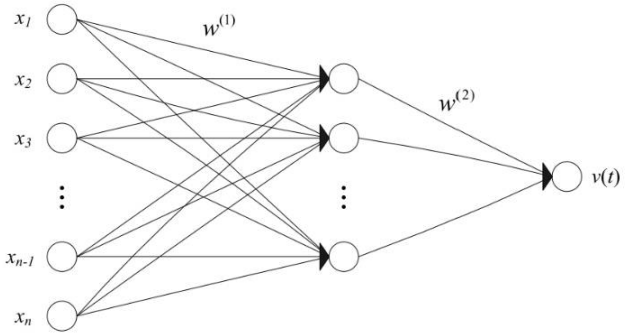

Figure 12: Critic Network

Besides, the output $v(t)$ of the neural network which indicates the predicted winning rate of the player with the certain chess board state is derived as follows, and as you can see, we choose the sigmoid function as our activation function.

$$h_i(t) = \sum_{j=1}^{n} x_j(t) w_{ji}^{(1)}(t) \quad g_i(t) = \frac{1}{1+\exp^{-h_i(t)}}$$

$$p(t) = \sum_{i=1}^{m} w_i^{(2)}(t) g_i(t) \quad v(t) = \frac{1}{1+\exp^{-p(t)}}$$

where $w_{ji}^{(1)}$ is the weight between j^{th} input node and the i^{th} hidden node; x_j is the j^{th} input of the input layer; n is the total number of input nodes; $h_i(t)$ is the input of the i^{th} hidden node; $g_i(t)$ is the output of the i^{th} hidden node; $w_i^{(2)}$ is the weight between hidden node and output node; m is the total number of hidden nodes; $p(t)$ is the input of the output node.

4.3 Feature Extraction

In our critic network, there are 81 nodes in the input layer, 32 nodes in the hidden layer and 1 node in the output layer. The input layer is consisted with 80 pattern features and 1 identity for the player (the

white or the black), while the 80 features are derived form the patterns we described above, there are 10 patterns in total (five, open-four, simple-four, etc.), which can be flattened to four-elements tuple in the following way.

Count of the pattern	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
> 3	1	1	1	1	$(\mathrm{n}-3)/2$

4.4 ADP with MCTS

Inspired by the paper, in order to get a more accurate prediction of winning probability, we calculate the weighted sum of ADP and its corresponding winning probability of MCTS. It is defined as:

$$w_p = \lambda w_1 + (1-\lambda) w_2$$

where w_{p} is the final winning probability of prediction, w_1 is the winning probability of the ADP, w_2 is the winning probability of the MCTS, λ is a real constant between $[0, 1]$. As it implies, when $\lambda = 0$, the winning prediction only depends on the MCTS. On the contrary, $\lambda = 1$ means that winning prediction only depends on the ADP.
The main pseudo code is presented below.

Algorithm 1: ADP with MCTS

input original state s_0;
output action *a correspond to ADP with MCTS*;
M_{ADP}, W_{ADP} ← ADP Stage(s_0);
W_{MCTS} ← MCTS Stage(M_{ADP});
for each w_1, w_2 in pairs(W_{ADP}, W_{MCTS}) **do**
  w_p ← $\lambda w_1 + (1\text{-}\lambda) w_2$;
  add p into P;
end for each
return action a correspond to max p in P

ADP Stage(state s)
  obtain top 5 winning probability W_{ADP} from ADP(s) ;
  obtain their moves M_{ADP} correspond to W_{ADP};
  return M_{ADP}, W_{ADP}

MCTS Stage(moves M_{ADP})
  for each move m in M_{ADP} **do**
    create m as root node with correspond state s
    obtain w_2 from $MTCS(m, s)$
    add w_2 into W_{MCTS}
  end for each
  return W_{MCTS}

5 PERFORMANCE

The best agent of all time, pyzobrist, deserved applause at around 1600 rating under Bayesian Elo.

-	YIXIN18	WINE18	pela	ZETOR17	pyzobrist	SPARKLE	EULRING16	NOESIS04	PISQ04	VALKYRIE13	PUREROCKY16	FIVEROW08	mushroom
YIXIN18	-	1 : 5	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6
WINE18	5 : 1	-	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6
pela	6 : 0	6 : 0	-	1 : 5	1 : 5	0 : 6	1 : 5	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6	0 : 2
ZETOR17	6 : 0	6 : 0	5 : 1	-	3 : 3	1 : 5	0 : 6	1 : 5	0 : 6	0 : 6	1 : 5	1 : 5	0 : 6
pyzobrist	6 : 0	6 : 0	5 : 1	3 : 3	-	1 : 5	1 : 5	2 : 4	1 : 5	0 : 6	0 : 6	1 : 5	0 : 6
SPARKLE	6 : 0	6 : 0	6 : 0	5 : 1	5 : 1	-	2 : 4	3 : 3	0 : 6	0 : 6	0 : 6	0 : 6	0 : 6
EULRING16	6 : 0	6 : 0	5 : 1	6 : 0	5 : 1	4 : 2	-	2 : 4	1 : 5	1 : 5	0 : 6	2 : 4	0 : 6
NOESIS04	6 : 0	6 : 0	6 : 0	5 : 1	4 : 2	3 : 3	4 : 2	-	3 : 3	0 : 6	0 : 6	0 : 6	1 : 5
PISQ04	6 : 0	6 : 0	6 : 0	6 : 0	5 : 1	6 : 0	5 : 1	3 : 3	-	1 : 5	0 : 6	4 : 2	1 : 5
VALKYRIE13	6 : 0	6 : 0	6 : 0	6 : 0	6 : 0	6 : 0	5 : 1	6 : 0	5 : 1	-	3 : 3	0 : 6	1 : 5
PUREROCKY16	6 : 0	6 : 0	6 : 0	5 : 1	6 : 0	6 : 0	6 : 0	6 : 0	6 : 0	3 : 3	-	1 : 5	0 : 6
FIVEROW08	6 : 0	6 : 0	6 : 0	5 : 1	5 : 1	6 : 0	4 : 2	6 : 0	2 : 4	6 : 0	5 : 1	-	2 : 4
mushroom	6 : 0	6 : 0	2 : 0	6 : 0	6 : 0	6 : 0	6 : 0	5 : 1	5 : 1	5 : 1	6 : 0	4 : 2	-
Total	71 : 1	67 : 5	53 : 15	48 : 24	46 : 26	39 : 33	34 : 38	34 : 38	23 : 49	16 : 56	15 : 57	13 : 59	5 : 63
Ratio	71.000	13.400	3.533	2.000	1.769	1.182	0.895	0.895	0.469	0.286	0.263	0.220	0.079
Points	36	33	30	25	25	19	18	14	10	7	7	6	0

As for ADP algorithm, limited by hardware facilities and training time, it didn't perform as well as expected, but with more training time comes a more intelligent agent.