ADP with MCTS Algorithm for Gomoku

Zhentao Tang, Dongbin Zhao, Kun Shao, Le Lv
The State Key Laboratory of Management and Control for Complex Systems. Institute of Automation, Chinese Academy of Sciences. Beijing 100190, China
tangzhentao2016@ia.ac.cn, dongbin.zhao@ia.ac.cn, shaokun2014@ia.ac.cn, iamlvle@126.com

***Abstract*—Inspired by the core idea of AlphaGo, we combine a neural network, which is trained by Adaptive Dynamic Programming (ADP), with Monte Carlo Tree Search (MCTS) algorithm for Gomoku. MCTS algorithm is based on Monte Carlo simulation method, which goes through lots of simulations and generates a game search tree. We rollout it and search the outcomes of the leaf nodes in the tree. As a result, we obtain the MCTS winning rate. The ADP and MCTS methods are used to estimate the winning rates respectively. We weight the two winning rates to select the action position with the maximum one. Experiment result shows that this method can effectively eliminate the neural network evaluation function's "short-sighted" defect. With our proposed method, the game's final prediction result is more accurate, and it outperforms the Gomoku with ADP algorithm.**

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

I. INTRODUCTION

Computer board games have been the focus of artificial intelligence research for a long time. Gomoku is a popular two-player strategical board game. It is traditionally played with Go pieces (black and white stone) on a board with 15x15 intersections. The winner is the player who first obtains an unbroken row of five pieces horizontally, vertically or diagonally. For solving such games, some typical methods were raised, such as the proof-number search [1], dependency-based search [2] and thread-space search [3]. And one of the most classic algorithms of playing Gomoku is the game tree searching, which is based on the min-max tree combined with a board evaluation function of leaf board situations. However, as William said [4], a complete search to a depth of n moves requires evaluations of $p!/(p-n)!$ board situations, where p is the current number of legal moves. Hence, to finish a search completely is an impossible task. Fortunately, the history heuristic with alpha-beta search has been used to speed up game tree search [5]. Although we all know that the deeper a solver can search in a game tree, the more effective it is. These methods have an obvious defect: time and space complexity growing exponentially with search depth. In other words, the depth of search can always be a bottleneck.

To solve this problem, we propose a new algorithm for Gomoku that combines shallow neural network with Monte Carlo simulation. Employing ADP to train the neural network and playing against itself can produce a professional player for Gomoku. After training, the neural network can get the wining probability of any possible board situation. Actually, we use neural network to evaluate board situations and obtain reasonable quantities of candidate moves to be taken. Then, we take these candidate moves as root nodes of MCTS and attempt to integrate our move prediction network with MCTS. Therefore, we obtain two results of winning probability respectively from neural network and MCTS. The final wining probability of prediction is the maximum sum in the weighted neural network and MCTS results.

The organization of the remaining paper is arranged as follows: in Section II, we discuss some related work using neural network or reinforcement learning for Gomoku. Section III provides a brief description of the MCTS. Section IV presents the implementation of ADP with MCTS in detail. Section V presents the experimental results that show the performance of ADP with MCTS algorithm and the compared results. Finally, we present discussion and summarize the paper with pointing out a direction for future research.

II. RELATED WORK

Early in 1990s, Freisleben proposed a neural network that had the capacity of learning to play Gomoku [6]. Its essence was to train a network by rewarding or penalizing from a special reinforcement learning algorithm, which was called comparison training [7]. Reinforcement learning is a novel machine learning method which concerns how software agent ought to take actions in an 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 Texas Hold'em Poker [8] and Go [9]. Furthermore, reinforcement learning has been used as a model for explaining the action-selection mechanism of the brain [10]. Temporal difference (TD) learning has primarily been used for the reinforcement learning problem, which is a prediction-based machine learning method. TD learning algorithm was applied to Gomoku by Mo [11] and Gong [12]. Nevertheless, the experiment results have shown that this approach for Gomoku is not as effective as TD-Gammon [13].

In spite of this, we think that in TD learning, the action decision or the value function can also be described in a continuous form, approximated by a nonlinear function line in neural networks. This is the core idea of Adaptive Dynamic Programing (ADP) [14-16]. The performance of the ADP for Gomoku has been improved by pairing it with a three-layer fully connected neural network to provide adaptive and self-teaching behavior. However, the input of the neural network was designed by pre-determined pattern. Therefore, the

This work is supported by National Natural Science Foundation of China (NSFC) under Grants No. 61273136, No.61573353 and No. 61533017

network was only effective for those games with available expert knowledge. Also, it has a short-sighted defect for neural network evaluation function.

Monte Carlo Tree Search (MCTS) is a method for finding optimal decisions in a given domain by taking random simulations in the decision space and building a search tree according to the results. Besides, it has a long history within the numerical algorithm and significant successes in various AI games, such as Scrabble [17], Bridge [18], especially for Go [19], like MoGo [20], ManGO [21]. Although MCTS was also introduced to Gomoku, it did not take a very good effect as expected. This is mainly because MCTS needs some complex domain knowledge additionally to work on a high level. Besides, MCTS must spend lots of time in simulation to get a satisfactory result.

The computer Go program AlphaGo, created by DeepMind, won 4:1 in a five game match against Lee Sedol, is one of the world's best Go player. According to the DeepMind's paper [22], AlphaGo uses a novel method combining deep neural network with the Monte Carlo simulation to evaluate board situation and selects the best move. Inspired by it, we apply Monte Carlo Tree Search into Gomoku, as well as combining with our previous work [23]. Accordingly, we actually obtain the final win rate both from ADP and MCTS algorithms.

III. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [24] requires a large number of simulation and builds up a large search tree according to the results. An important feature of MCTS is its estimated value will become more and more accurate with the increase of the simulation times and nodes accessed. The basic process of MCTS is shown in Fig. 1. It consists of four main stages: Selection, Expansion, Simulation, and Backpropagation.

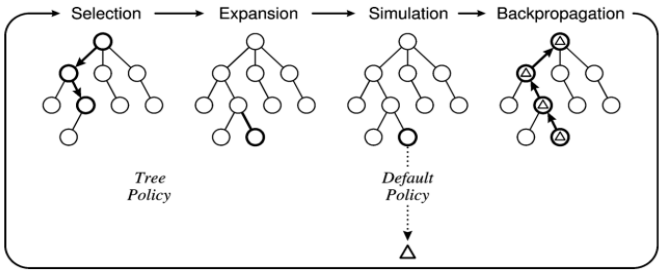

Fig. 1. The basic process of MCTS [24].

The basic process of MCTS starts from the first stage called Selection. At this stage, it 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. Then at Expansion stage, it can add one or more child nodes to expand the tree according to the available actions. At the third stage called Simulation, it can run a simulation from the leaf node based on the settled policy (or called Default Policy) to produce an outcome. Finally, at Backpropagation stage, it can back propagate the simulation result through the selected nodes to update their state values.

In this paper, we present two kinds of MCTS algorithms. One is called Heuristic Monte Carlo Tree Search (HMCTS), and the other is called Upper Confidence bounds for Tree (UCT). The HMCTS for Gomoku is presented in Algorithm 1.

Algorithm 1: HMCTS for Gomoku

input original state s_0;
output action a corresponding to the highest value of MCTS;
add Heuristic Knowledge;
obtain possible action moves M from state s_0;
for each move m in moves M **do**
  reward $r_{total} \leftarrow 0$;
  while simulation times < assigned times **do**
    reward $r \leftarrow$ Simulation($s(m)$);
    $r_{total} \leftarrow r_{total} + r$;
    simulation times add one;
  end while
  add (m, r_{total}) into *data*;
  end for each
return action Best(*data*)

Simulation(state s_t)
  if (s_t is win **and** s_t is terminal) **then return** 1.0;
                else return 0.0;
  end if
  if (s_t satisfied with Heuristic Knowledge)
    then obtain forced action a_f;
      new state $s_{t+1} \leftarrow f(s_t, a_f)$;
    else choose random action $a_r \in$ untried actions;
      new state $s_{t+1} \leftarrow f(s_t, a_r)$;
  end if
  return Simulation(s_{t+1})

Best(*data*)
  return action a //the maximum r_{total} of m from data

Note that here f is a function to generate a new board state from last board state and action. Heuristic knowledge which is common knowledge for Gomoku players can save more time in simulation than random sampling. Therefore, it helps the result getting converge earlier than before. The rules are explained as follows:

- If four-in-a-row is occurred in my side, the player will be forced to move its piece to the position where it can emerge five-in-a-row in my side.
- If four-in-a-row is occurred in opposite side, the player will be forced to move its piece to the position where it can block five-in-a-row in opposite side.
- If three-in-a-row is occurred in my side, the player will be forced to move its piece to the position where it can emerge four-in-a-row in my side.

- If three-in-a-row is occurred in opposite side, the player will be forced to move its piece to the position where it can block four-in-a-row in opposite side.

Though Gomoku is a zero-sum game like Go, a draw result rarely occurs in Gomoku. As a matter of fact, the final result usually turns out to be win or lose. Therefore, we make the reward be 1 when the final result is win or 0 when the final result is loss or a draw. Then the Q-value of an action can represent the expected reward of that action.

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{N(s)} l_i(s,a) z_i \quad (1)$$

where $N(s,a)$ is the number of times that action a has been selected from state s, $N(s)$ is the number of times that a game has been played out from s, z_i is the result of the i^{th} simulation played out from s, and $l_i(s,a)$ is 1 if action a is selected on the i^{th} playout from s or 0 otherwise.

The other widely used MCTS algorithm is UCT [24], which is based on Upper Confidence Bounds (UCB). UCB is known as capable to solve the multi-armed bandit problem. The most obvious virtue of UCB is that it helps to balance the conflict between exploration and exploitation and find out the final result earlier. Its simplest form is:

$$UCB = \bar{x}_j + \sqrt{\frac{2\ln(n)}{n_j}} \quad (2)$$

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 $\sqrt{2\ln(n)/(n_j)}$ encourages the exploration of less visited choices.

UCT is originated from HMCTS, but the difference to HMCTS is that the UCB can help to find out the suitable leaf nodes earlier than original algorithm, thus, UCT can save more time than the original version.

The UCT for Gomoku is presented in Algorithm 2.

Algorithm 2: UCT for Gomoku

input create root node v_0 with state s_0;
output action a corresponding to the highest value of UCT;
while within computational budget **do**
  $v_l \leftarrow$ Tree Policy(v_0);
  Policy $\leftarrow$ Heuristic Knowledge;
  reward $r \leftarrow$ Policy($s(v_l)$);
  Back Update(v_l, r);
end while
return action a(Best Child(v_0))

Tree Policy(node v)
  while v is not in terminal state **do**
    if v not fully expanded **then return** Expand(v);
      else $v \leftarrow$ Best Child(v, $1/\sqrt{2}$);
    end if
  end while
  return v //this is the best child node

Expand(node v)
  choose random action $a \in$ untried actions from $A(s(v))$;
  add a new child v' to v
    with $s(v') \leftarrow f(s(v), a)$ and $a(v') \leftarrow a$;
  return v' //this is the expand node

Best Child(node v, parameter c)
  return $\arg\max_{v' \in child}((Q(v')/N(v')) + c\sqrt{2\ln N(v)/N(v')})$

Policy(state s)
  while s is not terminal **do**
    if s satisfied with heuristic knowledge **then**
      obtain forced action a;
    else choose random action $a \in A$(s) uniformly;
    end if
    $s \leftarrow f(s, a)$;
  end while
  return reward for state s

Back Update(node v, reward r)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$;
    $Q(v) \leftarrow Q(v) + r$;
    $v \leftarrow$ parent of v;
  end while

Here, v indicates a node which has four pieces of data: the state $s(v)$, the next action $a(v)$, the total simulation reward $Q(v)$, the visited count $N(v)$. And v_0 is the root node corresponding to state s_0, v_l is the last node reaching the end of the game simulation, r is the reward for the terminal state reached by running the policy, the result of the overall search a(Best_Child(v_0)) is the action a that leads to the best child of the root node v_0.

Note that the MCTS is required to be repeatedly carried out for enough times to ensure the prediction can be more accurate. The most serious problem about time consuming in MCTS is that MCTS must spend a lot of time on searching some unnecessary feasible actions (unnecessary actions mean it wins in a low winning probability).

IV. Adaptive Dynamic Progamming with Monte Carlo Tree Search

Adaptive dynamic programming (ADP) used in Gomoku is trained by temporal difference learning (TDL), which is a

widely used reinforcement learning algorithm. The ADP training structure is illustrated in Fig. 2. The details for training the ADP can be seen in [23]. To solve the problem we have mentioned before, we try to obtain candidate action moves by ADP. Every one of candidate moves obtained from ADP should be the root node corresponding to each progress of MCTS. In other words, not only does it ensure the accuracy of the search, but also reduces the width of search. Compared with only using MCTS, it should save a large amount of time to find out the suitable action for Gomoku.

The current board state $x(t)$ is fed forward to the Action Selection, which generates the control action $u(t)$. Under the action $u(t)$, we obtain the next step transition state $x(t+1)$, which is fed forward to the utility function r which produces a reward $r(x(t+1))$. The critic network is used to estimate the cost function V. Then the reward $r(x(t+1))$, the estimate $V(t)$ and the estimate $V(t+1)$ are used to update the weights of the critic network to make the cost function V satisfy with the Bellman equation.

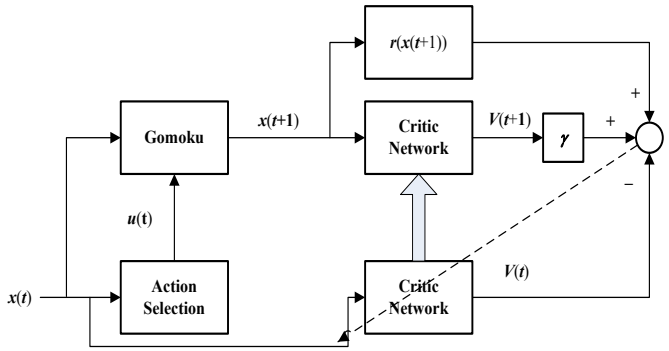

Fig. 2. The ADP structure.

The critic network in the ADP of Gomoku is a feed forward three-layer fully connected neural network. The structure is shown in Fig. 3.

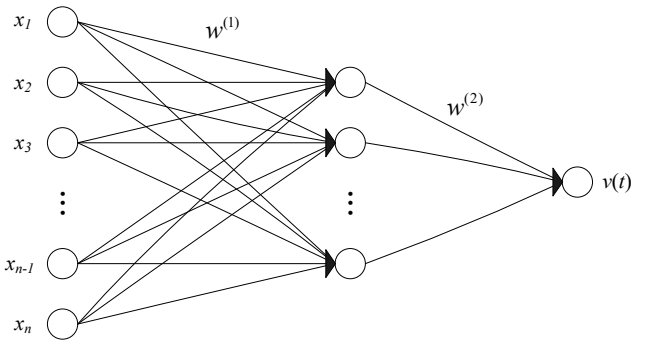

Fig. 3. The structure for the critic network [23].

The final output $v(t)$ of the neural network is the winning probability of the player with a board state, derived as follows.

$$h_i(t) = \sum_{j=1}^{n} x_j(t) w_{ji}^{(1)}(t) \tag{3}$$

$$g_i(t) = \frac{1}{1+\exp^{-h_i(t)}} \tag{4}$$

$$p(t) = \sum_{i=1}^{m} w_i^{(2)}(t) g_i(t) \tag{5}$$

$$v(t) = \frac{1}{1+\exp^{-p(t)}} \tag{6}$$

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;.

In the critic network, there are 274 nodes in the input layer, 100 nodes in the hidden layer and 1 node in the output layer. In the input layer, there are five input nodes indicating the number of every pattern except for the five-in-a-row. The coding method is shown in Table I. The reason why the number of input nodes is 274 is also mentioned in [23].

TABLE I. CODING METHOD OF THE INPUT NODES [23]

The number 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
4	1	1	1	1	0
>4	1	1	1	1	(n-4)/2

[a.] n is the number of one kind of input pattern.

As the experiment result shown in [23], we select the program ST-Gomoku [23], which has the best performance compared with the rest of cases, as our neural network evaluation function. But unlike our previous work, instead of only getting one move which has the maximum wining probability by neural network evaluation function, we will obtain 5 candidate moves with the top five wining probability.

Inspired by the idea of AlphaGo, we try to use ADP to train a shallow neural network combining with MCTS. Firstly, we obtain 5 candidate moves and their winning probabilities from the neural network which is trained by ADP. We call them ADP winning probabilities. Secondly, the 5 candidate moves and their situations of board are seen as the root node of MCTS. Then, we obtain 5 winning probabilities respectively from MCTS method. We call them MCTS winning probabilities. 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:

$$w_p = \lambda w_1 + (1-\lambda) w_2 \quad (7)$$

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 λ=0, the winning prediction only depends on the MCTS. On the contrary, λ=1 means that winning prediction only depends on the ADP.

The full ADP-MCTS is presented in Algorithm 3.

Algorithm 3: 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 ← $\lambda w_1 + (1-\lambda) w_2$;
  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}

Here s_0 indicates original state. M_{ADP} is the set of ADP's moves, and M_{MCTS} is the set of MCTS's moves. W_{ADP} is the set of top 5 winning probabilities by the ADP, and W_{MCTS} is the set of winning probabilities by the MCTS. ADP and MCTS Stages mean to find out their winning probabilities by the ADP or MCTS respectively from feasible moves.

As shown in Fig. 4, when λ=0.5 it seems to be the best to the final result. In other words, it appears only when its dependency to ADP and MCTS is balanced, the prediction of winning probability will be more accurate.

It should be pointed out that the candidate moves, obtained from ADP, make the MCTS's search space smaller than before. That is why ADP with MCTS saves more time than the method only uses MCTS. The other thing should be noted is the reason for just selecting 5 as the number of candidate moves. When the number of candidate moves is much bigger than 5, it will not save time as much as expected. In contrast, it most likely to

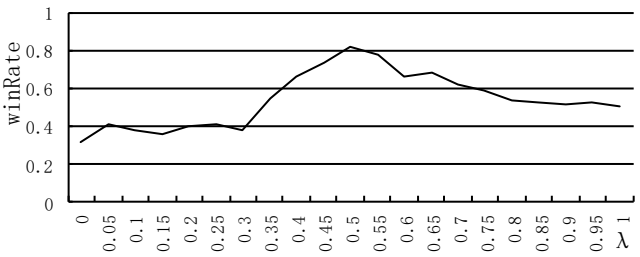

Fig. 4. Winning rate against ADP depends on λ.

be the same as the ADP if the number of candidate moves is smaller than 5. Also, 5 is based on the experiment's results, which turns out that 5 is a balance number for time consuming and playing level of Gomoku.

V. Experiments and Analysis

The present methods combining ADP with HMCTS or UCT both have been implemented. Our goals are as follows: first, compare the difference between HMCTS and UCT. Then, compare four different methods mentioned above and pick out the one which performs best when playing against each other. Finally, the best one, which is selected, will play against a commercial program called 5-star Gomoku [25].

In these experiments, the test system is based on a hardware platform of AMD A10-5750M APU with Radeon(tm) HD Graphics 2.50GHz while the software platform is Windows 10. Additionally, Open Multi-Processing (OpenMP) is an application programming interface that supports multi-platform shared memory multiprocessing programming in C and C++, thus, we use it to help MCTS convergence to the final result earlier.

A. Comparison between HMCTS and UCT

In Table II, the number in the first column and its corresponding rows represents the times of simulation with UCT or HMCTS. Considering the same MCTS method, the more simulations to be taken, the higher probability to win. This proves that the precision of winning probability could be improved with increasing the number of simulations. Note that the precision of MCTS prediction increases with a logarithmic form, hence, it will be almost imperceptible when the number of simulation is over 1,000.

TABLE II. Comparision Between HMCTS And UCT

HMCTS VS UCT	Score	Ratio
HMCTS-1 VS UCT-1	62:38	1.632
HMCTS-10 VS UCT-10	75:25	3.0
HMCTS-50 VS UCT-50	71:29	2.448
HMCTS-100 VS UCT-100	67:33	2.030
HMCTS-200 VS UCT-200	60:40	1.50
HMCTS-400 VS UCT-400	58:42	1.381

Table II lists HMCTS-*n* or UCT-*n*, where *n* represents the times of simulation board game. And when *n* is in a small number of simulation, HMCTS's performance is better than UCT's. However, with the simulation times increasing, the level of UCT will be closer to HMCTS. But overall, HMCTS's level is a little higher than UCT, while the cost time of HMCTS is less than UCT.

It is also worth noting that because of too much time consuming of MCTS, we limit the times of simulation to 400. In fact, when the times of simulation reach 400, it may cost 2 to 3 minutes to decide a suitable move, so it is obviously difficult to be used for Gomoku. In addition, it would not make significant progress when it only increases a little simulation times.

The worst thing about MCTS is the time consuming ensured to obtain a more accurate prediction. While, most of the feasible moves are totally valueless.

B. Comparison among four differnece methods

Note that we only use HMCTS as the MCTS to be the competitor, because it has been proved that HMCTS's performance is better than UCT's. Table IV shows that the levels for Gomoku from high to low are ADP-UCT, ADP-HMCTS, ADP, HMCTS.

TABLE III. COMPARISION AMONG 4 ALGORITHMS

Algorithm	Opponent Algorithm				Total	Ratio
	ADP-UCT	ADP-HMCTS	ADP	HMCTS		
ADP-UCT	-	267:233	360:140	397:103	1024:476	2.151
ADP-HMCTS	233:267	-	354:146	379:121	966:534	1.809
ADP	140:360	146:354	-	348:152	634:866	0.732
HMCTS	103:397	121:379	152:348	-	376:1124	0.335

Table III shows that ADP-MCTS's performance is better than ADP's and much better than MCTS's. Though UCT's performance is worse than HMCTS's, ADP-UCT's performance is a little better than ADP-HMCTS's.

During the experiment, the time consumed by the proposed methodology is mainly on MCTS. The cost in ADP is only about 80ms per move. Nevertheless, MCTS can spend much more time than ADP, which is around 5~10mins per move. The mainly reason for causing this time consumed is that MCTS may spend lots of time in simulations with a large number of possible moves. To make it effective, we use ADP to select the final candidate moves in a small amount (such as 5.), so it can guarantee the MCTS's simulation in a short time. The results show that the cost of ADP-HMCTS is reduced to 2~3*s* using OpenMP from originally 10*s* a move without it. However, ADP-UCT just spends about 4~5*s* a move without using OpenMP. Obviously, ADP-UCT not only stronger but also faster than ADP-HMCTS.

C. Playing against 5-star Gomoku

TABLE IV. COMPARISION AGAINST 5-STAR GOMOKU

Algorithm	Gomoku Level		
	Beginner	Dilettante	Candidate
ADP	100:0	73:27	43:57
HMCTS	46:54	13:87	0:100
ADP-HMCTS	100:0	89:11	71:29
ADP-UCT	100:0	82:18	64:36

Table IV indicates ADP-HMCTS and ADP-UCT both reach the level of Candidate, while ADP reaches the level of Dilettante and HMCTS nearly reaches the level of Beginner.

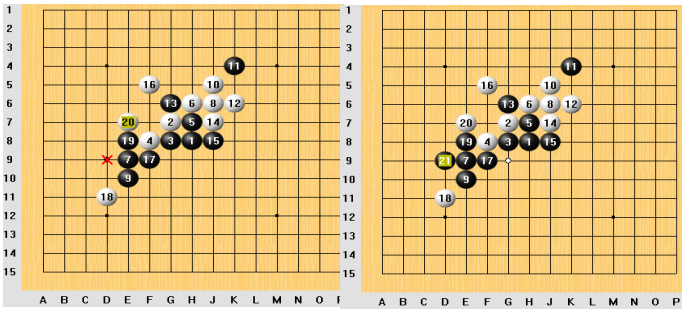

Fig. 5. How ADP-UCT to select its move in an informal game against ADP.

TABLE V. HOW ADP-UCT TO SELECT ITS MOVE IN AN INFORMAL GAME AGAINST ADP.

Five candidates of 21th move	ADP Prediction	UCT Prediction	Final Prediction
(H,9)	0.338	0.566	0.452
(F,7)	0.375	0.638	0.507
(E,11)	0.303	0.668	0.486
(G,10)	0.345	0.645	0.495
(D,9)	0.359	0.686	0.523

Table V shows that there are always five candidates of each move. The selected move is determined by the maximum of the final prediction value, which is equivalent to 0.5×(ADP Prediction+UCT Prediction). What we can see from Table V and Fig. 5. (D,9) is the final selected move which ensures to win. In this turn, it mainly depends on the UCT. While ADP is still in the position of the first two. The fact shows that the UCT may be the excellent supplement of the ADP, and it truly raises the accuracy of the prediction of winning probability. Actually, the ADP and UCT improve the performance of playing Gomoku.

VI. DISCUSSION AND CONCLUSION

In the previous studies of Gomoku, as an usual method, α-β pruning is used to generate node order, while the concrete operation in accessing the nodes is to compute the value of static evaluation function. The facts show that it has made a

big role in the traditional algorithm of Gomoku. While, this traditional method also brings 3 serious problems. The first is that static evaluation function always requires complicated artificial design and it needs a lot of time to consider plenty of situations. The second is that it can not learn anything while playing Gomoku. It just obeys the rule which is made before, and could not be improved by playing. The last is that the depth of search is always a bottleneck. The time and space complexities will grow exponentially with search depth, which limits the real-time performance of the game-tree-based solvers.

However, we can train a neural network that is able to learn to play Gomoku using ADP. And it has turned out that ADP's program for Gomoku approaches the candidate level of 5-star Gomoku. Meanwhile, it just costs one or two milliseconds for obtaining a prediction of winning probability for the neural network, and it can decide a move for average 60ms to 80ms. Thus, it looks to be much quicker than game-tree-based solvers. Moreover, we can train the neural network by playing against itself, and it shows its capacity to improve Gomoku level by learning through the situation of board. That is to say we can improve the level by training the neural network rather than programming with the rules.

Overall, we present a method by employing ADP combined with MCTS algorithm to solve a strategical game in this paper. From the experiment, ADP with MCTS has competed the candidate level of 5-star Gomoku. However, it still has a certain gap with YiXin, the best AI program for Gomoku. Although developing a stronger AI for a certain board game is the ultimate goal, self-playing is still a powerful technique as proven by AlphaGo. We will try to employ deep neural network to make a better feature representation of the board in the next stage.

ACKNOWLEDGMENT

We especially acknowledge Zhen Zhang and Yujie Dai for their valuable preliminary work. And we also thank DeepMind for their great work about AlphaGo.

REFERENCES

[1] L. V. Allis, M. V. D. Meulen and H. J. V. D. Herik, "Proof-number search," Artificial Intelligence, vol. 66, pp. 91-124, 1994.

[2] I. C. Wu, H. H. Kang, H. H. Lin, P. H. Lin, T. H. Wei, and C. M. Chang, Dependency-Based Search for Connect6: Springer International Publishing, 2013.

[3] L. V. Allis, H. J. Van Den Herik and M. P. H. Huntjens, "Go-Moku and Threat-Space Search," Interview Questions, 1994.

[4] W. T. Katz and S. Pham, "Experience-based learning experiments using Go-Moku," in IEEE International Conference on Systems, Man, and Cybernetics, 1991, pp. 1405-1410 vol.2.

[5] J. Schaeffer, "The history heuristic and alpha-beta search enhancements in practice," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 11, pp. 1203-1212, 1989.

[6] B. Freisleben, "A Neural Network that Learns to Play Five-in-a-Row," in New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, 1995. Proceedings, 1995, p. 87-87.

[7] G. Tesauro, "Connectionist learning of expert preferences by comparison training," in Advances in neural information processing systems 1, 1989, pp. 99-106.

[8] F. A. Dahl, "A Reinforcement Learning Algorithm Applied to Simplified Two-Player Texas Hold' em Poker," in Machine Learning: Emcl 2001, European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001, Proceedings, 2001, pp. 85-96.

[9] D. Silver, R. Sutton and M. Müller, "Reinforcement Learning of Local Shape in the Game of Go.," in Proceedings of the International Joint Conference on Artificial Intelligence, Hyderabad, India, January, 2007, pp. 1053-1058.

[10] F. Ishida, T. Sasaki, Y. Sakaguchi, and H. Shimai, "Reinforcement-learning agents with different temperature parameters explain the variety of human action-selection behavior in a Markov decision process task," Neurocomputing, vol. 72, pp. 1979-1984, 2009.

[11] J. W. Mo, "Study and practice on Machine Self-Learning of Game-Playing.,". vol. Master Thesis: Guangxi Normal University, 2003.

[12] R. M. Gong, "Research and Implementation of Computer Game Strategy Based on Reinforcement Learning,". vol. Master Thesis: Shenyang Ligong University, 2011.

[13] G. Tesauro, "Temporal difference learning and TD-Gammon," Communications of the ACM, vol. 38, pp. 58-68, 1995.

[14] A. G. Barto, R. S. Sutton and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," IEEE Transactions on Systems Man & Cybernetics, vol. SMC-13, pp. 834-846, 1983.

[15] J. W. Paul, "A menu of designs for reinforcement learning over time," Neural networks for control, MIT Press, Cambridge, MA, 1990.

[16] D. Zhao, Z. Xia, D. Wang, "Model-free optimal control for affine nonlinear systems based on action dependent heuristic dynamic programming with convergency analysis," IEEE Transactions on Automation and Science Engineering. vol. 12, no. 4, pp. 1461–1468, 2015

[17] A. Ramírez, F. G. Acuña, A. G. Romero, R. Alquézar, E. Hernández, A. R. Aguilar, and I. G. Olmedo, "A Scrabble Heuristic Based on Probability That Performs at Championship Level," MICAI 2009: Advances in Artificial Intelligence, vol.5845 Lecture Notes in Computer Science, pp. 112-123, 2009.

[18] M. L. Ginsberg, "GIB: Imperfect information in a computationally challenging game," Journal of Artificial Intelligence Research, pp. 303-358, 2001.

[19] A. F. Smith and G. O. Roberts, "Bayesian computation via the Gibbs sampler and related Markov chain Monte Carlo methods," Journal of the Royal Statistical Society. Series B (Methodological), pp. 3-23, 1993.

[20] S. Gelly and Y. Wang, "Exploration exploitation in go: UCT for Monte-Carlo go," in NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop, 2006.

[21] G. M. J. Chaslot, M. H. Winands, H. J. V. D. HERIK, J. W. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," New Mathematics and Natural Computation, vol. 4, pp. 343-357, 2008.

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no.7587, pp. 484-489, 2016.

[23] D. Zhao, Z. Zhang and Y. Dai, "Self-teaching adaptive dynamic programming for Gomoku," Neurocomputing, vol. 78, pp. 23-29, 2012.

[24] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, pp. 1-43, 2012.

[25] L. Atomax, "http://www.5-star-gomoku.com-about.com/", 2006.