IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 16, 2019, accepted August 24, 2019, date of publication August 28, 2019, date of current version September 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2937943

A Middle Game Search Algorithm Applicable to Low-Cost Personal Computer for Go

XIALI LI[1], ZHENGYU LV[1], SONG WANG[1], ZHI WEI[2], XIAOCHUAN ZHANG[3], AND LICHENG WU[1]

[1]School of Information Engineering, Minzu University of China, Beijing 100081, China
[2]Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA
[3]School of Artificial Intelligence, Chongqing University of Technology, Chongqing 401135, China

Corresponding author: Licheng Wu (wulicheng@tsinghua.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61602539, Grant 61873291, and Grant 61773416, and in part by the MUC 111 Project.

ABSTRACT Go Artificial Intellects(AIs) using deep reinforcement learning and neural networks have achieved superhuman performance, but they rely on powerful computing resources. They are not applicable to low-cost personal computer(PC). In our life, most entertainment programs of Go run on the general PC. A human Go master consider different strategies for different stages, especially for the middle stage that has a significant impact on winning or losing. To study arguably a more humanlike approach that is applicable to low-cost PC while not reducing chess power, this paper proposes a new search algorithm based on hypothesis testing and dynamic randomization for the middle stage of the game Go. Firstly, a new method to decide the intervals of different playing stages more reasonable based on hypothesis testing is proposed. Secondly, a new search algorithm including a layered pruning branch method, a comprehensive evaluation function and a new selecting node method is proposed. The pruning method based on domain knowledge and upper confidence bound formula(UCB) are all applied to subtract the branches from the lower evaluation score, which was ranked behind 20%. The comprehensive evaluation function with adjustable parameters is proposed to evaluate the tree nodes after pruning. The new selecting node method based on dynamic randomization is used to expand the tree by selecting a node randomly from the high-quality node interval. Finally, the experimental results show that the designed algorithm outperforms Gnugo3.6 and Gnugo3.8 in chess power while reducing average search time and average RAM cost for one move effectively on a 19×19 board.

INDEX TERMS Go, search algorithm, MCTS, UCT, hypothesis test, dynamic randomization.

I. INTRODUCTION

Currently, deep learning is successfully utilized in many feilds [1], [2]. Deep reinforcement learning and neural networks have been used in Go research and have achieved superhuman performance. Alpha Zero, AlphaGo Zero, and Alpha Go are the state-of-art programs for the game of Go [3]–[6]. Alpha Zero, which was proposed in 2018, leverages a general reinforcement learning algorithm that enables it to master Go, chess, and shogi through self-play. A total of 5000 first generation tensor processing units (TPUs) were used to generate the self-play games, and 16 second-generation TPUs were used to train the neural networks of AlphaZero [3]. A multi-core central processing unit(CPU) was used to train the network in AlphaGo Zero [4]. A total of 1920 CPUs and 280 graphics processing units (GPUs) were used in the distributed system of AlphaGo [5], [6]. Clearly, AIs based Go programs such as Alpha Zero, AlphaGo Zero, and Alpha Go all need huge computation resources and hardware support. This holds true for other AIs based Go programs as well; for instance, a total of 2000 GPUs were used to train ELF OpenGo [7], which is an open source Go AI. With only several multi-core CPUs, the computation and chess power of DeepZenGo are far below those of AlphaGo [8]. LeelaZero is a civilian-level Go AI that requires the concerted efforts of players all over the world to provide massive simulation games for its growth. Currently, there are over 2,000 machines participating in the LeelaZero project and the computing power is not high [9], [10].

The associate editor coordinating the review of this article and approving it for publication was Guan Gui.

Deep reinforcement learning applied in Go relies on powerful high-performance computing resources to compliment the self-play and training the deep neural networks. The superman Go programs run on common configuration desktop computer seldom. But in our life, most entertainment Go programs need to run on the general PC. Thus, it is practical for researchers to study the search algorithm applicable to low-cost personal computer. Upper confidence bound apply to tree (UCT) algorithm, Monte Carlo tree search (MCTS) algorithm and varieties based on them are the typical representatives suitable for running on the common configuration personal computer.

A more humanlike approach to searching is attractive [11]. A human Go master will apply different strategies at different stages of the game and situations during the whole game. For example, a formalized series of moves are used in the opening stage, and attacking and defending strategies are employed in the middle game. Designing different algorithms for different stages of Go is arguably a more humanlike approach. But most of UCT or MCTS based algorithms ignore the effect of playing stage on the performance of the game. And there is no special research on search algorithm for middle stage although which is the longest and most important for the whole Go game.

A multi-modal search algorithm with thresholds for different stages based on experimental observations for Go whole game had been proposed. Pattern recognition and matching methods were used for the starting and ending stages, and MCTS algorithm was used for the middle stage [12]. However, the thresholds decision method lacks statistical data support.

In this paper, we first conducted the special research on search algorithm for Go middle stage. Hypothesis testing was firstly applied in Go research to obtain more reasonable intervals. The middle stage of Go search algorithm is based on hypothesis testing and dynamic randomization to approach the goal of low computation and no loss of winning rate for the low-cost personal computers.

The proposed algorithm has three main compositions: a layered pruning branch method, a comprehensive evaluation function with adjustable parameters, and a new selecting node method based on dynamic randomization. The pruning methods based on domain knowledge and UCB formula are all applied to subtract the branches with the lower evaluation score, which were ranked behind 20%. The evaluation function based on domain knowledge and the UCB formula is proposed to evaluate the tree nodes after pruning. The parameters of the functions are adjustable by experiments. A random node in the high-quality node interval with the top 15% of the comprehensive evaluation scores is selected and used to expand the tree.

The experiments demonstrate that our proposed novel middle game algorithm can achieve higher winning rates with the low time and space consumption, running on a normal personal computer configuration, compared with Gnugo3.6 and Gnugo3.8. The main contributions of this work can be summarized as follows: Firstly, it is the first time to engage in designing a searching algorithm for the middle stage of Go. This work is of significant importance as the middle stage lasts longest and is usually critical in the outcome of the game.

Secondly, it is the first time that a hypothesis test method is applied to Go to determine the threshold interval for the middle stage of Go. This provides a reasonable and statistical length episode for using the proposed middle stage algorithm.

Furthermore, the layered pruning method based on domain knowledge and UCB formula is simple and effective in reducing search time and storage, while retaining the chess power. The pruning method can reduce approximately 17 percentage points of average search time and approximately 10 percentage points on average RAM cost for one move, compared to Gnugo3.8 on a 19×19 board.

Finally, the parameters of the evaluation function can be adjusted feasibly by experiments. Besides, selecting a node in the high-quality node interval randomly to extend the tree not only improves the chess power, but also adds diversity while avoiding falling into a local optimum.

The rest of this paper is organized as follows. Related works are briefly reviewed in Section 2. Our proposed search algorithm is described in detail in Section 3. In Section 4, experimental results and analysis are presented. Finally, conclusions and future works are summarized in Section 5.

II. RELATED WORKS

As most Go algorithms that do not rely on high computing resources employ upper confidence bound apply to tree (UCT) algorithm, Monte Carlo tree search (MCTS) algorithm or their varieties, this paper give a brief review of UCT algorithm, MCTS algorithm, pruning methods, and other related works associated with UCT or MCTS algorithm in this section.

UCT algorithm uses UCB formula and Monte Carlo simulation to prune the game tree and evaluate the situation of leaf nodes, respectively [13]–[15]. It also deeply explores well-performing nodes. Gelly et al. leveraged the time difference algorithm to speed up the search of UCT trees,however, the knowledge gained from online learning often shows large deviations [16]. He et al. proposed a knowledge-based information ceiling tree for node evaluation of UCT algorithm [17]. The All moves as first (AMAF) is an enhancement closely related to the history heuristic first proposed in Monte Carlo Go. By adopting a small number of simulated rounds, AMAF method was used to speed up the evaluation of the situation. However, the evaluation method was completely randomized [18]. AMAF-caused deviation was corrected by adjusting parameters. This correction significantly improved the gaming power of Go program. However, this method exists the searching depth threshold [19].

MCTS is currently classical computer game method for finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results [20], [21]. MCTS has already had a profound impact on AI-based approaches that can be

represented as trees of sequential decisions, particularly games. Many variations and enhancements for MCTS have been developed [22], [23]. Chen et al. applied domain knowledge and dynamic randomization to improve MCTS algorithm by randomizing the parameters in selected ranges during the in-tree phase of a simulation game, and hierarchically randomizing move-generators during the play-out phase.

Dynamic randomization was used in [24]. It can increase the playing strength of a Go Intellect significantly while the simulations are beyond 128K per move. GPU-based go search method was developed by Zhang et al. This schema can improve the searching speed of MCTS algorithm. However, it lacks the suitable searching algorithm to fit this scheme [25]. Fu et al. exploited a chessboard recognition algorithm based on chessboard image, however, the image quality had a significant effect on the effect of the algorithm [26]. A 9×9 board Go system based on time difference algorithm was designed in [27]. However, when the game was in the middle stage, the offensive ability of the system was clearly weakened. Guo et al. proposed a degenerated strategy to reduce the complexity of Go, however, how to prove the game after degradation equivalent to the original game is an unsolved problem [28].

Currently, game search tree pruning algorithms include soft pruning, progressive pruning, absolute pruning, relative pruning, and domain-knowledge-based pruning [22]. Soft pruning is not applicable to Monte Carlo search tree [29]. Based on Indigo's 9×9 board Go and 19×19 board Go, progressive pruning improves the search speed, however, it has little effect on gaming power [30]. Both absolute pruning and relative pruning can improve search efficiency. Relative pruning was applied in Lingo, and it improved the winning rate against Gnugo 3.8 by approximately 3%. Domain-knowledge-based pruning can significantly improve the winning rate of Lingo against Gnugo 3.8. The use of UCB-based pruning can also improve chess power [31].

The hypothesis test is used to observe whether significant differences occur using the mathematical distribution analysis and the sample observations obtained by sampling. It first proposes a hypothesis of the proposition being studied - the hypothesis that there is no significant difference- before using a certain method to verify whether the hypothesis is true [32]. Hypothesis testing theory has been successfully applied to quality system certification [33], image resolution criteria [34], indoor multi-feature detection of mobile robot [12]. However, it has not been applied to Go.

From the above literature, we can see that most Go algorithms based on UCT or MCTS do not consider the playing stage effect on the outcome to the game. There is no special middle stage algorithm for Go game. The threshold decision method lacks statistical data support although the multi-modal algorithm for different stages has been proposed.

To make the Go program think more like a human with higher chess power and lower space and time resource, we designed a novel algorithm with a more reasonable interval decision method, which focuses on the middle stage of Go running on a personal desktop computer. The hypothesis test method is firstly applied to determine the threshold interval for middle stage of Go. This method is of statistical significance. The algorithm consists of a layered pruning method, a comprehensive evaluation function, and the new randomly dynamic selecting node method. The algorithm can improve the game power of the program more than pattern matching and UCT algorithms applied to Gnugo3.6 and Gnugo3.8. Experiments demonstrate that the proposed algorithm achieves desirable performance in improving chess power and reducing RAM and search time resources.

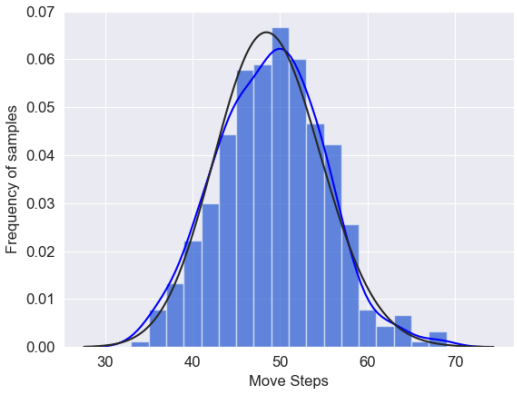


FIGURE 1. Step threshold statistical histogram into middle stage.

III. PROPOSED SEARCH ALGORITHM FOR MIDDLE STAGE

In this section, we first provide the threshold interval method using the hypothesis test. Then we describe the algorithm that uses the comprehensive evaluation function, the pruning method, and node selecting method in detail.

A. THRESHOLD INTERVAL METHOD FOR THE NUMBER OF STEPS OF MIDDLE-STAGE GO GAME

We obtained 500 games from a publicly accessible Go website [35].These games were played by Zhou Ruiyang, Li Changyi, Coulee L and Li Shishi before May 2015.

A 5th dan Go player, Ya Dong, from our research team analyzed the 500 games record and extracted the raw data of the number of steps at which the game enters the middle stage and end stage respectively for each game. 450 valid data points in the steps of the game entering the middle game and 335 valid data points in the steps entering the end game are valid from the 500 games record. We use discrete random variables X and Y to represent the critical steps to enter the middle game and the end game respectively. We got the statistical histogram diagrams of the threshold steps at which Go game enters the middle and end games respectively. The outcome is illustrated in Fig.1 and Fig.2. The ordinate indicates the number of samples and abscissa indicates the move steps in Fig.1 and Fig.2. From Fig.1 and Fig.2, it can be seen that the threshold steps X and Y follow an approximately normal distribution. Therefore, it is reasonable and practical applying hypothesis test method to verify the distribution

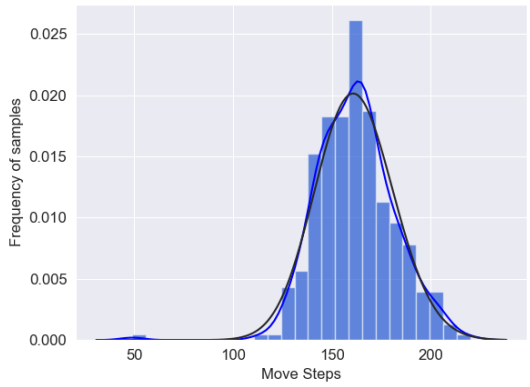


FIGURE 2. Step threshold statistical histogram into final stage.

of the steps. If a random variable Z follows the normal distribution, its probability density function is expressed by the equation 1.

$$g\left(z,\mu,\sigma^2\right)=\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{1}$$

The likelihood function of the mean and variance of with m samples is expressed by equation 2.

$$\begin{aligned}L=L\left(\mu,\sigma^2\right)&=\prod_{i=1}^{m}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2\sigma^2}(z_i-\mu)^2}\\&=\left(2\pi\sigma^2\right)^{-\frac{m}{2}}e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{m}(z_i-\mu)^2}\end{aligned} \tag{2}$$

According to the sample size, Z is divided into j sub-intervals $[b_i, b_{i+1})$ $(i = 1, 2, \ldots j)$ that do not overlap with each other. Let $B_i = \{b_i \le z \le b_{i+1}\}$ and H_0 be the hypothesis. If H_0 is true, the distribution function of Z is

$$\begin{aligned}G_O(z)&=\int_{-\infty}^{z}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(t-\mu)^2}{2\sigma^2}}dt=\Phi\left(\frac{z-\mu}{\sigma}\right),\\\Phi(z)&=\int_{-\infty}^{z}\frac{1}{\sqrt{2\pi}}e^{-\frac{t^2}{2}}dt\end{aligned} \tag{3}$$

The classical goodness of fit test is [36] divide the entire possible randomized trial into incompatible events $B_1, B_2, \ldots, B_k (\bigcup_{i=1}^{k} B_i = O, B_i B_j = \varnothing, i \ne j,$ $i, j = 1, \cdots, k)$. Generally, if H_0 is true and the number of tests is large, the statistic of H_0 is represented by the equation 4.

$$\chi^2=\sum_{i=1}^{k}\frac{(g_i-mP_i)^2}{mP_i}\left(\chi^2=\sum_{i=1}^{k}\frac{\left(g_i-m\hat{P}_i\right)^2}{m\hat{P}_i}\right) \tag{4}$$

If sample number m is large enough, $g\left(z,\mu,\sigma^2\right)$ follows an approximately χ^2 distribution with $k-r-1$ degrees of freedom. The goodness of fit reflects the degree of conformity between the actual data and the theoretical model. When the goodness of fit is less than or equal to the significance level α, i.e., the χ^2 obtained by equation 1 satisfies the inequality

$$\chi^2 \ge \chi^2_{1-\alpha}(k-r-1) \tag{5}$$

reject H_0; otherwise accept H_0. The probability of this test being the first type of error is approximately α[32]. Suppose X and Y are subject to normal distribution. The sample size of X is 450 and the sample size of Y is 335. We take the parameter estimation process of X as an example. Let $\alpha = 0.05$, from equation 1 we have $\chi^2_{1-\alpha}(k-r-1) = \chi^2_{1-0.05}(8-2-1) = \chi^2_{0.95}(5) = 11.071$. since $\chi^2 = \sum_{i=1}^{k}\frac{(g_i-mP_i)^2}{mP_i}\left(\chi^2=\sum_{i=1}^{k}\frac{\left(g_i-m\hat{P}_i\right)^2}{m\hat{P}_i}\right) = 1.4440$ We have $\chi^2 = 1.4440 < 11.071$. H_0 is accepted under level 0.05. The number of steps in which Go enters the middle stage follows a normal distribution is verified.

Considering the first type of error, let $\alpha = 0.01$ or $\alpha = 0.05$. The hypothesis H_0 is proven to be true. Thus, the threshold steps at which Go game enters the middle stage follows $X \sim N\left(\mu_1, \sigma_1^2\right)$ which satisfies $\mu_1 = 49$ and $\sigma_1 = 6$. The threshold steps Y follows $Y \sim N_2\left(\mu_2, \sigma_2^2\right)$ which satisfies $\mu_2 = 162$ and $\sigma_2 = 19$ by the above same method. Let T_1 and T_2 be the threshold intervals of entering the middle game and end game, respectively. We then have the following:

$$T_1 = [\mu_1 - \sigma_1, \mu_1 + \sigma_1] = [43, 55] \tag{6}$$

$$T_2 = [\mu_2 - \sigma_2, \mu_2 + \sigma_2] = [143, 181] \tag{7}$$

Let T be the threshold interval of the middle game. Thus, there is

$$T = \left[t_1, t_2\right] \tag{8}$$

where t_1 and t_2 are any two elements, each selected from sets T_1 and T_2, respectively.

Thus, the threshold interval of the middle game based on hypothesis test is obtained using the above procedures. A certain value in this interval set can be selected as critical step randomly. This step provides a reasonable and statistical length episode for using the proposed middle stage algorithm.

B. DESCRIPTION OF MIDDLE-STAGE ALGORITHM FOR GO

The proposed algorithm framework is based on MCTS. It has three main parts: a layered pruning branch method, a comprehensive evaluation function with adjustable parameters, and a new node selecting method based on dynamic randomization. The pruning method is used to subtract the branches from the lower evaluation score, which was ranked behind 20% when extending the tree. The comprehensive evaluation function is then used to evaluate the node of the tree. Dynamic randomization selecting is applied to select one node randomly from the high quality interval with the top 15% of the comprehensive evaluation scores. Assume X represent the number of steps in Go. When $t_1 \le X \le t_2$, repeat the steps of the following pseudocode (See Algorithm 1).

1) LAYERED PRUNING METHOD

Searching lots of nodes in the game tree needs a large of time and space resources. Pruning methods are very nec-

Algorithm 1 Middle-Stage Algorithm for Go

Set the current situation as the root node of the search tree S_{root};
while *timeremain* **do**
    Extend the tree by MCTS for the root node S_{root};
    Prune the branches using the proposed PDKAUCT;
    Evaluate the remain nodes using the proposed comprehensive evaluation function based on domain knowledge and the UCT algorithm;
    Select a node S randomly as the target move using the proposed RSHQI method;
    return S;
    Replace S_{root} with S;
end while

Algorithm 2 Pruning Methods

function Pruning()
    for $i = 0 \rightarrow n$ **do**
        Evaluate node i using domain knowledge;
        Output evaluated value set $d(d_1, d_2, \ldots, d_n)$;
        Sort by descending d;
        Pruning 20% nodes ranked at the bottom of set d;
        Return evaluated value of the rest 0.8n nodes;
    end for
    for $i = 0 \rightarrow 0.8 \times n$ **do**
        Evaluate node i using the UCT alogrithm with UCB1 formula;
        Output evaluated value set $u(u_1, u_2, \ldots, u_n)$;
        Sort by descending u;
        Pruning 20% nodes ranked at the bottom of set u;
        Return evaluated value of the rest 0.8*0.8n nodes;
    end for
end function

essary to subtract nodes with poor performance and can improve the search efficiency to some extent. Pruning methods applied in the go game searching algorithm include domain-knowledge-based pruning, progressive pruning, soft pruning, relative pruning, absolute pruning, etc. [31]–[33]. Domain knowledge based pruning methods and UCT based pruning methods are effective currently among the different methods. They can not only improve the search efficiency but also contribute more to the gaming power. Therefore, we proposed a layered pruning method that evaluate the nodes before subtracting. It is composed by two sequent steps.

First, evaluate the nodes according to domain knowledge and subtract the branches with the low value score ranked at the bottom 20%. Second, evaluate the remained nodes according to UCB1 [37] and subtract the branches with the low value score ranked at the bottom 20%. 20% is selected as the pruning ratio according to Pareto's principle. The pruning methods is described in the following pseudocode (See Algorithm 2).

TABLE 1. Different weight distribution schema of domain knowledge and UCT evaluation.

schema	UCT weight	domain knowledge weight
A	0.5	0.5
B	0.4	0.6
C	0.3	0.7
D	0.2	0.8
E	0.1	0.9

2) COMPREHENSIVE EVALUATION FUNCTION

A comprehensive evaluation function with dynamic adjusted parameters combining domain knowledge and UCT algorithm is described in detail in this section. The parameters were adjusted according to test data.

For a node i in the game tree, the static evaluation value of its domain knowledge is denoted as $d_i(0 \leq i \leq 360)$ while the UCB1 evaluation value is represented by $u_i(0 \leq i \leq 360)$. The evaluation value of the current game V_i can be formulated as follows:

$$V_i = f(\mathrm{t}) \times d_i + g(\mathrm{t}) \times u_i \tag{9}$$

where $f(\mathrm{t})$ and $g(\mathrm{t})$ are weight function of domain knowledge evaluation and UCB1 evaluation. They are adjustable parameters, satisfying $0 \ll f(t) \leq 1$, $0 \ll g(t) \leq 1$, and $f(t) + g(t) = 1$. Pruning method based on domain knowledge performs better than other pruning methods in improving chess power [22], [31]. Therefore, the weight of domain knowledge evaluation is greater than that of the UCT evaluation, i.e., $f(t) \geq g(t)$. The weight values are adjusted according to actual test results of the go program. Suppose $f(t) = (0.5, 0.6, 0.7, 0.8, 0.9)$. According to the five suits of UCB1 evaluation weight and domain knowledge evaluation weight, five different weight distribution schema, A, B, C, D, and E, are listed in Table 1. Five different weight distribution schema were applied to develop 19×19 board Go game respectively. In order to select a better weight distribution scheme, the five programs are compared using a circular game. One program plays 100 games against another for each pair of programs. The compassion of winning rates with different weight distribution schema are shown in Fig.3.

From Figure 3, it can be seen that B weight distribution scheme, which is with the highest winning rate 0.545, is much better than other categories. Therefore, B scheme is selected as the parameter set of the comprehensive evaluation function.

3) NODE SELECTING METHOD

The moves with the maximum winning expectation at some certain steps may not indicate a final victory in subsequent games. In this paper, a high-quality node interval with the top 15% evaluation scores is constructed. One node located in the high-quality interval is selected randomly rather than the node with the highest score. The process of node selecting

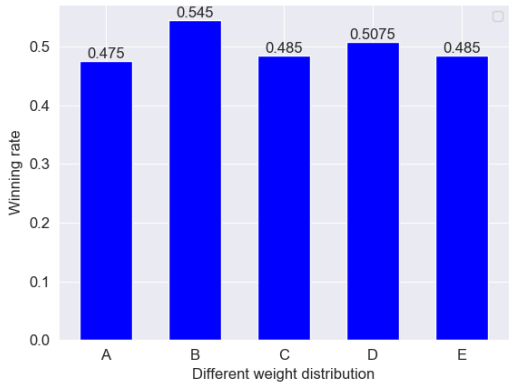


FIGURE 3. Winning rates comparison among five schema.

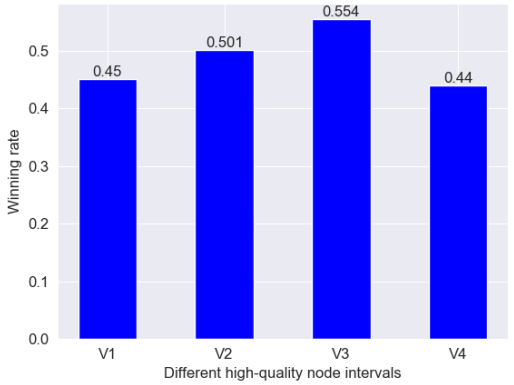


FIGURE 4. Winning rates comparison among different high-quality node intervals.

method is in the following. First, sort the child nodes of the current situation in descending order of their evaluation weights. Second, construct the high-quality node interval range with evaluation weights in top 15% range. Finally, a node is randomly selected from this interval to perform game tree expansion. The range of the high-quality interval 15% was determined according to actual test data. During the test, we set V1, V2, V3, and V4 to be the ranges schema with the top 5%, 10%, 15%, and 20% evaluation scores respectively. The four different high-quality node interval schema were applied in the 19×19 board Go respectively. Experiments were conducted to select the optimal scheme by playing circular games among the programs using V1, V2, V3, and V4. A total of 100 games were played between each pair of programs. The winning rates of different schema are shown in Fig.4. It can be seen that V3 with the highest the highest winning rate 0.554 is the best schema from Fig.4. Therefore, the high-quality node interval with the top 15% evaluation scores was selected as the target range.

IV. EXPERIMENT RESULTS AND ANALYSIS

We developed the search program named Mucgo for three categories boards using the proposed algorithm based on hypothesis test and dynamic randomization for the middle stage of go game. In order to verify the performance of this new algorithm, we conducted experiments extensively to compare chess power and computation consumption of the proposed Go program with those of Gnugo3.6 and Gnugo3.8. There are some difference for different boards of Mucgo. The proposed algorithm was used for all stages in 9×9 and 13×13 boards. But in 19×19 board, the proposed algorithm was used for the middle stage, while a pattern-matching method based on domain knowledge was used for the start and final stages. Gungo3.6 uses a pattern-matching method based on domain knowledge, whereas Gnugo3.8 uses a pure UCT algorithm without pruning branches. The main methods and algorithm applied in the three programs are listed in Table.2. Mucgo, Gnugo3.6 and Gnugo3.8 were deployed on a normal configuration desktop computer. Mucgo played against Gnugo3.6 and Gnugo3.8 automatically on the GoGui gaming platform. 9 and 13 board programs were running on one computer and 19 board programs were running on the other computer. Details on the running environment for these Go programs of different board sizes are given in Table.3.

TABLE 2. Methods applied in Mucgo, Gnugo3.6 and Gnugo3.8.

	Mucgo	Gnugo3.6	Gnugo3.8
Pruning method	PDKAUCT	No	No
Evaluation method	Domain knowledge, UCT	No	UCT
Node selection	RSHQI	No	UCT

TABLE 3. Hardware configuration of program running environment.

	9×9,13×13 boards	19×19 board
Processor	Inter(R)Core(TM) i3-2120 CPU@3.30GHZ	Inter(R) Core(TM) i5-2400 CPU@3.10GHZ
Memory	2G	4G
Operating system	32-bit Windows7 Ultimate operating system	64-bit Windows7 Professional operating system

A. CHESS POWER EVALUATION COMPARED WITH GNUGO3.6 AND GNUGO3.8

Automatic gaming experiments on the two desktop computers lasted approximately 1 month. A tournament was played to evaluate the chess power of Mucgo compared to that of Gnugo3.6 and Gnugo3.8; the results of this tournament are illustrated in Fig.5 and Fig.6. In the top bar, Mucgo plays against specialized programs on a 9×9 board; in the central bar, Mucgo plays against specialized programs on a 13×13 board; and in the bottom bar, Mucgo plays against specialized programs on a 19×19 board. Each bar shows the results from Mucgo's perspective: win (W; green), draw (D; gray), or loss (L; red). The left part in Fig.5 shows the wins, draws, and losses of Mucgo on 9×9, 13×13, and 19×19 boards when playing against Gnugo3.6. The win, draw, and loss rates are 52.42, 21.32, and 26.26 percentage

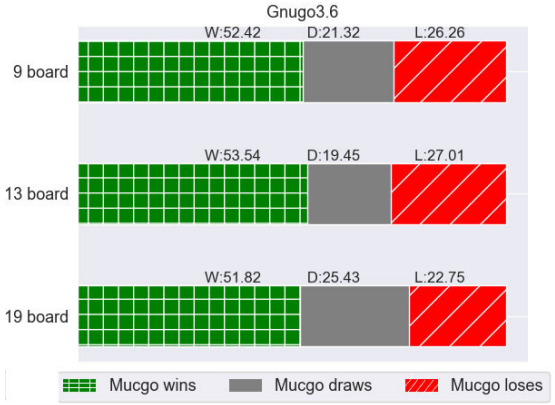


FIGURE 5. Chess power comparison with Gnugo3.6.

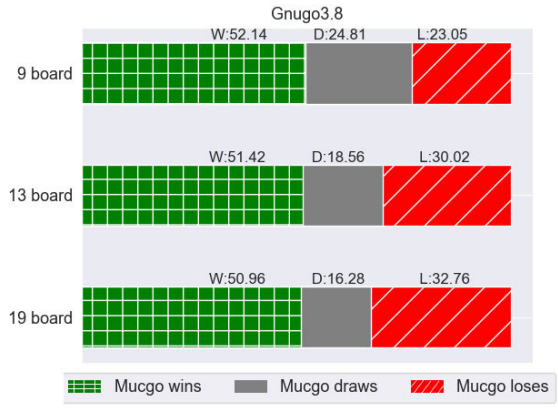


FIGURE 6. Chess power comparison with Gnugo3.8.

points on a 9×9 board; 53.54, 19.45, and 27.01 percentage points on a 13×13 board; and 51.82, 25.43, and 22.75 percentage points on a 19×19 board, respectively. It can be seen that the winning rate is higher than 50 percentage points and the loss rate is below 35 percentage points. As Mucgo is a program developed using the proposed MAHTADR algorithm, which includes a layered pruning method, a comprehensive evaluation function, and a new node selecting method, and Gnugo3.6 is a program developed using a pattern-matching method based on domain knowledge, Mucgo performs better than Gnugo3.6 when chess power is considered. The experiments verify that the proposed middle stage algorithm can improve the chess power effectively when compared with the pattern-matching algorithm used in Gnugo3.6. The automatic game experiments between Mucgo and Gnugo3.8 on the two desktop computers lasted approximately 15 days. A total of 891 valid and complete 9×9 board game records, 1195 valid and complete 13×13 board game records, and 970 valid and complete 19×19 board game records were collected. The right part in Fig.6 shows the wins, draws, and losses of Mucgo on 9×9, 13×13, and 19×19 boards when playing against Gnugo3.8. The win, draw, and loss rates are 52.14, 24.81, and 23.05 percentage points on a 9×9 board; 51.42, 18.56, and 30.02 percentage points on a 13×13 board; and 50.96, 16.42, and 32.76 percentage points on a 19×19 board, respectively. Evidently, Mucgo has a higher winning rate and a lower loss rate against Gnugo3.8. Gnugo3.8 is a program developed using the UCT algorithm and it includes no pruning method or improvements. In contrast, Mucgo is developed using the proposed algorithm with considerable improvements; thus, it outperforms Gnugo3.8 when considering chess power. The experiments verify that the proposed algorithm performs better than the UCT algorithm used in Gnugo3.8 in terms of chess power.

1) SEARCH TIME AND RAM COST COMPARISON WITH GNUGO3.8

To verify the performance of the proposed algorithm in terms of reducing search time and RAM costs, we compared the average search time and RAM cost of one move

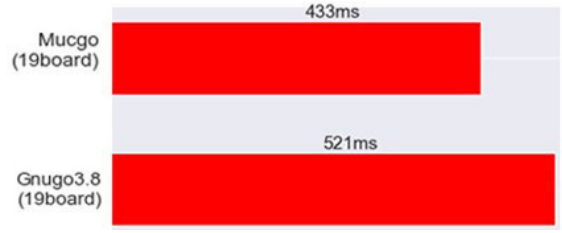


FIGURE 7. Average search time per move.

FIGURE 8. Average RAM cost per move.

of Mucgo and Gnugo3.8. The result is shown in Fig.7 and Fig.8. The search time and storage consumption is very low for the algorithm without using deep neural network. These parameters are of no obvious significance on the 9×9 and 13×13 boards because the search space is not large enough. Therefore, we only conducted experiments comparing Mucgo and Gnugo3.8 on a 19×19 board. In the first game, Gnugo.8 played as black and Mucgo played as white. The two programs played 215 moves in total to the outcome of the game. In the second game, Gnugo3.8 played as white and Mucgo played as black. They played 209 moves in total to the outcome of the game. For each move, the search time and RAM cost were recorded. The average search time and RAM cost was determined by calculating the mean values of all the moves for both Mucgo and Gnugo3.8.

In Fig.7, the red bars represent the average search time, which was 433 ms and 521 ms for Mucgo and Gnugo3.8, respectively. Mucgo reduced the average search time by approximately 17 percentage points compared to

Gnugo3.8.In Fig.8, the blue bars represent the average RAM cost for one move, which was 212 MB and 235 MB for Mucgo and Gnugo3.8, respectively. Mucgo reduced the average RAM cost by approximately 10 percentage points compared to Gnugo3.8. As Mucgo used a layered pruning method based on domain knowledge and the UCT algorithm, it could effectively reduce the search time and RAM cost compared to Gnugo3.8, which did not prune the search tree. These experiments verify that the proposed pruning method is very effective in saving both time and space resources.

V. CONCLUSION

Go game is generally divided into layout, mid-game and final stage, and the mid- game has a great influence on the outcome. Therefore, this paper considered the game progress and specially designed a search algorithm on the base of hypothesis testing and dynamic randomization for the middle stage of Go. The 500 game data of Go players Coulee, Li Shishi, Li Changhao and Zhou Ruiyang were analyzed. The fit goodness test and the maximum likelihood estimation method were used to verify the distribution of the threshold interval steps at which the game enter middle stage. We got the threshold interval steps at which game enter final stage by the same method. Thus, the threshold interval of the middle stage was gotten. The proposed algorithm framework has three main parts: a layered pruning branch method, a comprehensive evaluation function with adjustable parameters, and a new selecting node method based on dynamic randomization. The layered pruning method based on domain knowledge and UCT was used to subtract the branches with the lower evaluation score which were ranked behind 20%. To evaluate the tree nodes after pruning, a comprehensive evaluation function was proposed. The parameters of the evaluation function were adjusted dynamically, and the parameters with better performance were selected according to the experimental results. This is practical to improve the chess power. To expand the tree, a new selecting node method was given in this paper. Instead of selecting the node with the highest comprehensive evaluation score, select the high-quality node interval with the comprehensive evaluation score in the top 15%, and dynamically select one node from this interval to expand. This 15% high-quality node interval was selected as a better-performing interval according to experimental data. The algorithm was applied to design and implement 9×9, 13×13, and 19×19 board Go,named Mucgo. Automatic game experiments were conducted extensively with different versions (Mucgo, Gnugo3.6, and Gnugo3.8) for three Go categories of 9×9, 13×13, and 19×19 boards respectively.

The experiments verify that the proposed middle stage algorithm based on the hypothesis theory and dynamic randomization can improve the chess power effectively when compared with the pattern-matching algorithm used in Gnugo3.6. It also performs better than the UCT algorithm used in Gnugo3.8 in terms of chess power. It can effectively reduce the search time and RAM cost compared to Gnugo3.8.

Some research problems still need to be addressed. For example, the illustration of the reasonable pruning percentage, the existence of optimal parameters for the comprehensive evaluation function, and the existence of an optimal high-quality node interval. In the future, research on a general Go artificial intelligence model with light weight deep neural network structure, which considerably shortens training time and requires almost no human knowledge should be conducted.

ACKNOWLEDGEMENT

Songting Deng made some experiments. Yongji Li fufiled some programming work.

REFERENCES

[1] H. Huang, S. Guo, G. Gui, Z. Yang, J. Zhang, H. Sari, and H. Adachi, "Deep learning for physical-layer 5G wireless techniques: Opportunities, challenges and solutions," *IEEE Wireless Commun.*, to be published.
[2] H. Huang, Y. Song, J. Yang, G. Gui, and F. Adachi, "Deep-learning-based millimeter-wave massive MIMO for hybrid precoding," *IEEE Trans. Veh. Technol.*, vol. 68, no. 3, pp. 3027–3032, Mar. 2019.
[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, and T. A. Lillicrap, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018.
[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[6] 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.
[7] Y. Tian and Y. Zhu, "Better computer go player with neural network and long-term prediction," 2015, *arXiv:1511.06410*. [Online]. Available: https://arxiv.org/abs/1511.06410
[8] F. Ya. (2016). *Deepzengo Claimed to Defeat Alphago*. [Online]. Available: https://www.leiphone.com/news/201611/YwcDpatPjff5xxEa.html
[9] X. Lu and Q. Zhang. (2018). *Yuandong Tian, Member of Facebook, Open Source of the ELF Opengo*. [Online]. Available: https://www.jiqizhixin.com/articles/2018-05-03-7
[10] Y. Wei. (2018). *Website Built for Leela*. [Online]. Available: https://hhpetra.github.io/leelachinese/d_wv=1031
[11] C. E. Shannon, "Programming a computer for playing chess," in *Computer Chess Compendium*. New York, NY, USA: Springer, 1988, pp. 2–13.
[12] X. Li and L. Wu, "A multi-modal searching algorithm in computer go based on test," in *Proc. Chin. Intell. Automat. Conf.* Berlin, Germany: Springer, 2015, pp. 143–149.
[13] S. Gelly and Y. Wang, "Exploration exploitation in go: UCT for Monte-Carlo go," in *Proc. Neural Inf. Process. Syst. Conf. On-Line Trading Explor. Workshop*, 2006, pp. 1–9.
[14] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. Eur. Conf. Mach. Learn.* Berlin, Germany: Springer, 2006, pp. 282–293.
[15] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.
[16] S. Gelly and D. Silver, "Combining online and offline knowledge in UCT," in *Proc. 24th Int. Conf. Mach. Learn.*, 2007, pp. 273–280.

[17] S. He, Y. Wang, F. Xie, J. Meng, H. Chen, S. Luo, Z. Liu, and Q. Zhu, "Game player strategy pattern recognition and how UCT algorithms apply pre-knowledge of player's strategy to improve opponent AI," in *Proc. Int. Conf. Comput. Intell. Modelling Control Automat.*, Dec. 2008, pp. 1177–1181.
[18] D. P. Helmbold and A. Parker-Wood, "All-moves-as-first heuristics in monte-carlo go," in *Proc. IC-AI*, 2009, pp. 605–610.
[19] S. Gelly and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer go," *Artif. Intell.*, vol. 175, no. 11, pp. 1856–1875, 2011.
[20] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *Proc. Int. Conf. Comput. Games*. Berlin, Germany: Springer, 2006, pp. 72–83.
[21] G. Gui, H. Sari, and E. Biglieri, "A new definition of fairness for non-orthogonal multiple access," *IEEE Commun. Lett.*, vol. 23, no. 7, pp. 1267–1271, May 2019.
[22] 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 Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.
[23] G. Gui, H. Huang, Y. Song, and H. Sari, "Deep learning for an effective nonorthogonal multiple access scheme," *IEEE Trans. Veh. Technol.*, vol. 67, no. 9, pp. 8440–8450, Sep. 2018.
[24] K.-H. Chen, "Dynamic randomization and domain knowledge in Monte-Carlo tree search for go knowledge-based systems," *Knowl.-Based Syst.*, vol. 34, pp. 21–25, Oct. 2012.
[25] Q. F. Zhang, "Research and implementation of the parallel 9 × 9 go game engine based on cuda architecture," M.S. thesis, School Softw., Beijing Univ. Posts Telecommun., Beijing, China, 2012.
[26] S. B. Fu, "Research on the image recognition of go competition on the handheld mobile terminal platform," M.S. thesis, School Elect. Sci. Technol., Nanjing Univ., Nanjing, China, 2012.
[27] Y. Tang, "Research and application of machine learning algorithm for go computer games," M.S. thesis, School Comput. Sci. Eng., Chongqing Univ. Technol., Chongqing, China, 2012.
[28] J. Guo, "Research on several upper limit values in go artificial intelligence," M.S. thesis, School Math. Statist., Central South Univ., Changsha, China, 2013.
[29] C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong, "The computational intelligence of MoGo revealed in Taiwan's computer go tournaments," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 1, pp. 73–89, Mar. 2009.
[30] B. Bouzy, "Move-pruning techniques for Monte–Carlo go," in *Advances in Computer Games*. Berlin, Germany: Springer, 2005, pp. 104–119.
[31] J. Huang, Z. Liu, B. Lu, and F. Xiao, "Pruning in UCT algorithm," in *Proc. Int. Conf. Technol. Appl. Artif. Intell.*, Nov. 2010, pp. 177–181.
[32] J. Jin and Y. Su, "An improved adaptive genetic algorithm," *Comput. Eng. Appl.*, vol. 41, no. 18, pp. 64–69, 2005.
[33] W. M. Bian, "The application of hypothesis test in quality system audit," *Chem. Ind. Times*, vol. 18, no. 4, pp. 52–54, 2004.
[34] W. W. Wang and W. Zhengming, "New criteria and arithmetic for calculating the resolution in sar images based on hypothesis test," *Signal Process.*, vol. 24, no. 5, pp. 853–858, 2008.
[35] *YiCheng Weiqi Wang*. Accessed: Oct. 15, 2018. [Online]. Available: http://www.eweiqi.com/
[36] Y. J. Cai, "Note on two types of errors of statistical hypothesis tests," *Appl. Statist. Manage.*, vol. 18, no. 3, pp. 30–35, 1999.
[37] Z. Q. Liu and W. F. Li, *Foundation of Modern Computer Go Games*. Beijing, China: Beijing University of Posts and Telecommunication, 2011.

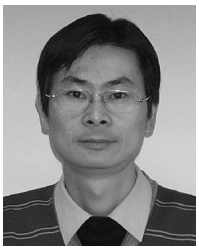
XIALI LI was born in Henan, China, in 1979. She received the M.S. degree in computer science and technology from Xi'an Jiaotong University (XJTU), Xi'an, China, in 2004. From October 2008 to August 2009, she was a Visiting Scholar with The University of Edinburgh, U.K. Her research interests include computer games and artificial robotics. She is the author of three books and more than 50 articles, and she holds more than two inventions.

ZHENGYU LV was born in Guiyang, Guizhou, China, in 1992. He received the bachelor's degree in engineering from the Minzu University of China, in 2015, where he is currently studying with the Modern Education Technology. His research interests include computer game and artificial robotics.

SONG WANG was born in Baoding, Hebei, China, in 1994. He received the bachelor's degree in engineering from Beijing Union University, in 2017, where he is currently studying with the Modern Education Technology. His research interests include computer game and artificial robotics.

ZHI WEI was born in Guangxi, China. He received the B.S. degree from Wuhan University, China, and the Ph.D. degree from the University of Pennsylvania, in 2008. He is currently an Associate Professor with the Department of Computer Science, New Jersey Institute of Technology. His research interests include data mining, statistical modelling, and machine learning with applications to data enriched fields.

XIAOCHUAN ZHANG was born in Sichuan, China, in 1965. He received the M.S. degree from Chongqing University (CQU), Chongqing, China, in 1991. He is currently a Full Professor and the Deputy Dean with the School of Artificial Intelligence, Chongqing University of Technology (CQUT). He is also the Chairman of Computer Game Professional Committee of CAAI. He is the author of three books, more than 90 articles, received four provincial and ministerial level awards for science and technology and teaching, and holds more than three inventions. His research interests include computer games, intelligent robot, and software engineering.



LICHENG WU was born in Jiangxi, China, in 1972. He received the bachelor's degree from the Beijing University of Aeronautics and Astronautics (BUAA), Beijing, China, in 1995, and the Ph.D. degree in robotics from the Institute of Robotics, BUAA, in 2000. In 2001 and 2002, he held a postdoctoral position with the State Key Laboratory of Intelligent Technology and Systems, Department of Computer Science and Technology, Tsinghua University, where he was promoted as an Associate Professor, in 2008. He was with the School of Information Engineering, Minzu University of China, in November 2009, where he was promoted as a Full Professor, in January 2013. From November 2006 to November 2007, he was a Visiting Scholar with Polytechnic di Milano and Cassino University, Italy. He is currently a Full Professor and the Dean of the School of Information Engineering, Minzu University of China, Beijing. He is the author of three books and more than 80 articles, and he holds more than three inventions. His research interests include artificial intelligence, computer game, and robotics.