IPSJ SIG Technical Report Vol.2020-GI-44 No.6
2020/6/27

An Experimental Evaluation of PUCT Algorithm with Convolutional Neural Network Evaluation Functions

Lucien Troillet[1,a)] Kiminori Matsuzaki[1,b)]

Abstract: One of the most successful Monte-Carlo tree search (MCTS) applications is AlphaGo and its successors in which neural network evaluation functions are combined with a variant of MCTS called PUCT (Predictor + UCB applied to trees). However, further investigation on how various factors (e.g., evaluation functions, reinforcement learning, number of simulations) impact its performance is still required. To answer this question, our previous work examined the impact of pattern-based (linear) evaluation functions on PUCT outcomes. In this paper, we try to analyze the PUCT algorithm with evaluation functions based on (non-linear) neural networks. We developed several convolutional neural networks attempting to replicate evaluation values of Zebra (an open-source champion-level player) with different quality. Through experiments feeding these to the PUCT algorithm, we find that it still consistently performs better than the evaluation function alone. It also appears that its behavior is independent of linear or non-linear evaluation function usage.

Keywords: Monte-Carlo tree search, PUCT, Convolutional Neural Networks, Analysis

1. Introduction

Monte-Carlo Tree Search (MCTS) [2], [4] is a game-playing method used to find an optimal move by performing random simulations of games called *playouts* and building a search tree according to the results. After the first proposal by Coulom [4], MCTS has been intensively studied for many games, especially Go. In theory, the upper confidence bounds applied to trees (UCT) algorithm [8] is proven to return an optimal move after an infinite number of playouts. In practice, however, we are required to select a move using a limited amount of time and resources.

Therefore, many strong MCTS players perform tree traversals and/or playouts with some game-specific heuristic bias. Such classical approaches include node prior [3], [5], progressive bias [6], and early playout termination (EPT) [7], [10]. EPT was successfully used in several players such as Amazons [7], [9], Breakthrough [11] and Havannah [10]. A more recent and successful approach is PUCT (Predictor + UCB applied to trees) [15]. The DeepMind team adopted a variant of the PUCT algorithm combined with deep neural networks and reinforcement learning techniques to develop very strong game players such as AlphaGo [17], AlphaGo Zero [19] and its successors [18], [23].

Intuitively, the evaluation functions used in these algorithms are desired to be more accurate in order to obtain stronger MCTS players. The extreme success of AlphaGo and AlphaGo Zero is said to come from accurate evaluation functions based on deep neural networks, but it remains unclear how much the PUCT algorithm contributes to the strength of players, especially with respect to the accuracy of evaluation functions. The nonlinear properties of neural networks might also be a key to the success of the PUCT algorithm in the AlphaGo series.

In our previous studies [12], [13], [16], we analyzed the effect of evaluation functions on the performance of MCTS algorithms using Othello (Reversi) positions as testing cases. We used the evaluation function of a champion-level open-source Othello player, Zebra [1], as the baseline. The evaluation function calculates an evaluation value by looking up local feature values and summing them up. We generated variants in a qualitative manner (by removing patterns) [12] and in a quantitative manner (by adding noise) [16]. The experiment results showed that we can improve the players' strength by the AlphaGoZero's PUCT algorithm independently from the evaluation function being good or bad. The limitation in our previous studies, however, lies in the fact that the evaluation functions used were inherently linear.

In this study we will perform a similar analysis using convolutional neural networks as evaluation functions. We design simple feed-forward convolutional networks with different convolution kernels, depth and number of parameters. The networks are trained in a supervised learning manner: they replicate the values of the evaluation function of Zebra or those after alpha-beta search using Zebra. We then test these networks in combination with AlphaGoZero's PUCT algorithm by changing the number of simulations.

Important findings in this study are summarized as follows.

- PUCT always improves upon the performance of the evaluation function but stagnates or improves very slowly after a certain number of simulations.
- For small numbers of simulations, most of the improvement in performance comes from the tree search being done, in-

[1] Kochi University of Technology, Kami, Kochi, 782–8502, Japan
[a)] 246005s@gs.kochi-tech.ac.jp
[b)] matsuzaki.kiminori@kochi-tech.ac.jp

dependently of how good the evaluation function is.

- We do not find significant influence of using linear or non-linear evaluation functions on the performance of PUCT.

The rest of the paper is organized as follows. In Section 2, we introduce the PUCT algorithm and present our approach to the performance evaluation of our players. In Section 3, we detail the design and training of our neural network evaluation functions. In Section 4, we present our experimental results and analyze them. We review related work in section 5 and conclude the paper in Section 6.

2. Preliminaries

2.1 AlphaGoZero's PUCT Algorithm

Similar to general Monte-Carlo tree search (MCTS) algorithms, AlphaGoZero's PUCT algorithm performs a number of simulations. The main difference with other MCTS algorithms resides in the use of two evaluation functions, policy and value functions, in the simulations. Note that the algorithm used in this study is based on the one used in AlphaGo Zero [19], and it is different from the original PUCT algorithm [15] and from the one used in AlphaGo [17]. The most important difference is in the evaluation phase below, where the result of the value function becomes an action value directly, without performing any playouts.

Let each edge (s, a) in the search tree store the average action value $Q(s, a)$, visit count $N(s, a)$, and prior probability $P(s, a)$. Each simulation of AlphaGoZero's PUCT algorithm consists of the following phases.

Selection Traverse the search tree from the root down to a leaf. Here, a child is selected at each step based on the average action value and bonus:

$$a_t = \underset{a}{\operatorname{argmax}} \left(Q(s, a) + c \frac{P(s, a)}{1 + N(s, a)} \right)$$

Here, c is a constant to adjust the exploration.

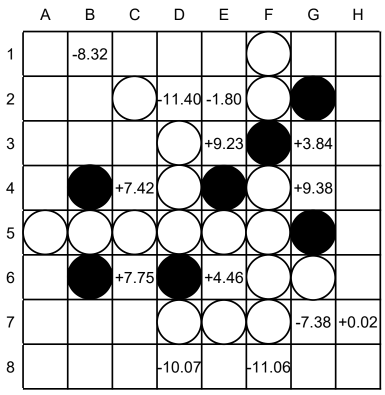
Expansion The leaf node may be expanded and children are added to the search tree. In this study, we expand the leaf if it was already traversed once.

Evaluation Let the resulting action value of the simulation be the evaluation value $V(s)$ of the leaf state s.

Backup The average action values and the visit counts on the path to the root are updated.

After a number of simulations, the algorithm chooses the most visited move at the root.

The evaluation value $V(s)$ and prior probability $P(s, a)$ are computed once when a leaf node is expanded. For the evaluation value $V(s)$, we simply used the result of evaluation functions in Section 3. For the prior probability $P(s, a)$, we used softmax of the evaluation values after the move a. The evaluation function of Zebra and thus those of neural networks in Section 3 return a value between -64×128 and 64×128, corresponding to the score at end of the game. Considering these large values from evaluation functions, we used the exploration constant $c = 1000$. Also, evaluation values are scaled by 0.0001 before being fed to the softmax function.



move	score	error
G4	+9.38	0
E3	+9.23	0.15
C6	+7.75	1.63
C4	+7.42	1.96
E6	+4.46	4.92
G3	+3.84	5.54
H7	+0.02	9.36
E2	−1.80	11.18
G7	−7.38	16.76
B1	−8.32	17.7
D8	−10.07	19.45
F8	−11.06	20.44
D2	−11.40	20.78

Fig. 1 Example position in T-20 with black as the next to move.

2.2 Test Positions and Evaluation Measures

Often in game programming, players are evaluated in comparison to other players through a series of matches (e.g., by Elo rating). This evaluation, however, requires a variety of players as well as long computation time. Another approach to evaluate players is focusing on the moves for a rather small set of positions [21].

In this study, we take the second approach and borrow the test positions and evaluation measures from our previous work [12]. Three sets of positions are used: 100 positions after 20 moves (T-20), 100 positions after 30 moves (T-30), and 100 positions after 40 moves (T-40). Each of these positions were analyzed using WZebra with a lookahead of 24 moves (therefore, the analyzed results are exact for T-40). We associated the score (the difference in the number of stones) for each legal move (Fig. 1). Note that very one-sided positions (with score of best move smaller than -12 or larger than $+12$) were excluded from the 100 positions.

To analyze the performance of our PUCT player and evaluation functions we utilize the following two measures.

Error Rate Let *error* be defined as the difference of scores between the best move and the selected move. *Error rate* is the average of errors of selected moves (the best is zero).

Best-move Ratio The ratio (in percent) that the selected moves are the best moves (the best is 100). When a move has ties, we count any with the best score as the best move.

3. Design of Evaluation Functions

In this study, we use the evaluation function of Zebra, an open-source top-level computer player, as a basis to train new evaluation functions. These functions are made using convolutional neural networks with varying depths, number of parameters and convolutional matrix shapes. They are given a valid Othello board as input and return a single value corresponding to the evaluation of the given board.

We trained a total of fifteen different Neural networks to replicate the evaluation values of four training data sets, generating 60 different neural network evaluation functions.

3.1 Network Models

Each network model is characterized by a convolution kernel size, a number of convolutional layers, a number of fully connected layers and a number of outputs. Combining these numbers

Table 1 Models and parameter sets

				epoch used (error rate for T-20) for training data from lookahead d			
Model	Parameter Set	Number of Filters	Number of weights	d=0	d=2	d=4	d=6
	Small	8, 16, 128	131993	50 (2.367)	50 (4.964)	43 (5.660)	43 (6.038)
$C_2 2F2$	Medium	32, 64, 256	2106977	26 (1.780)	41 (2.079)	31 (2.576)	28 (2.673)
	Large	128, 256, 2048	33692033	50 (1.345)	50 (2.011)	15 (2.336)	18 (2.632)
	Small	8, 16 (×3), 128, 32	80761	36 (2.782)	32 (4.905)	50 (8.160)	47 (3.385)
$C_2 4F3$	Medium	32, 64 (×3), 512, 128	1287649	31 (1.848)	29 (2.690)	22 (3.360)	47 (2.328)
	Large	128, 256 (×3), 2048, 512	20584321	12 (1.456)	46 (1.920)	39 (1.584)	50 (1.605)
	Small	8, 16 (×7), 128, 32	19385	50 (4.329)	50 (4.688)	46 (3.923)	21 (4.957)
$C_2 8F3$	Medium	32, 64 (×7), 512, 128	304865	36 (2.107)	46 (2.290)	23 (2.697)	49 (2.663)
	Large	128, 256 (×7), 2048, 512	4856705	12 (2.352)	30 (1.784)	16 (1.915)	38 (1.882)
	Small	8, 16, 128	75449	40 (1.936)	20 (4.674)	44 (3.816)	8 (3.570)
$C_3 2F2$	Medium	32, 64, 256	1200353	21 (2.208)	37 (4.752)	37 (2.531)	28 (2.994)
	Large	128, 256, 2048	19178369	43 (2.104)	50 (2.019)	50 (2.566)	32 (1.970)
	Small	8, 16 (×3), 128, 32	18585	50 (2.997)	50 (4.472)	42 (4.485)	26 (6.515)
$C_3 4F3$	Medium	32, 64 (×3), 512, 128	290913	48 (1.771)	23 (2.668)	24 (2.780)	22 (2.588)
	Large	128, 256 (×3), 2048, 512	4628865	12 (2.244)	28 (1.995)	46 (1.797)	39 (1.612)

we get a unique identifier for each network model. For example: a network using 3x3 kernels with 4 convolution layers, 2 fully connected layers and one output would have the following characteristic numbers : 3x3,4,2,1.

Given the nature of the task the neural network evaluation functions were designed to fulfill, they are all regressive and thus only have one output. We can therefore omit the last number as it is always the same. Moreover only kernels with the same height and width were used so we can omit the second kernel dimension as well. With this in mind, the whole structure of our example model can be expressed as $C_3 4F3$. $C_3 4$ describes the convolutional layers and $F3$ describes the the 2 hidden fully connected layers as well as the output layer with one neuron.

While we tested many different network models, we selected the 5 presented in Table 1.

3.2 Network Parameters

For each network model we used 3 different parameter sets which describe the number of channels on each convolutional layer or the number of neurons on each fully connected layer. Continuing the previous example of the $C_3 4F3$ model, we have a total of 6 layers before our unique output (4 convolutional and 2 fully connected). This means we need 6 different parameters to describe the number of channels/neurons of each layer. One parameter set for example could be 8,16,16,16,128,32.

Each of the parameter sets for a model has numbers of channels/neurons on each layer scaled by a factor of 4 times the values of the preceding parameter set. For future references in this paper all parameter sets of a model type are named Small(S), Medium(M) and Large(L). All specific parameter numbers are given in Table 1.

3.3 Training Data

To train our networks we generated a total of 1048576 board states by having computer players of various strength play against each other. All board states are then shuffled and evaluated by using the Zebra evaluation function with different lookaheads. In our case we used lookaheads of zero, two, four and six moves to have four different board sets with evaluations of varying strength.

3.4 Network Epoch Selection

All neural network evaluation functions were trained during a total of 50 Epochs. After this training period, in order to avoid overfitting of the networks we plotted the error rate of the single use of the evaluations functions on the T-20 set. We used these plots to visualize around which period each network consistently performed well and selected the best epoch from this period.

4. Experiment and Evaluation

4.1 PUCT Performance to Single-use Evaluation Function

We ran experiments to compare the performance of PUCT using Zebra's evaluation function and the neural-network evaluation functions we created. For each function, we ran the PUCT until it reached 128000 simulations for each position in T-20, T-30, and T-40. During these runs we saved the selected move at various intermediate numbers of simulations.

Tables 2 and 3 show the experiment results with less simulations (1000) and more simulations (128000) as well as the results of single use of evaluation function. Table 2 shows the cases when we used the evaluation values after alpha-beta with lookahead 0 (Zebra's evaluation function) as the training data. Table 3 shows the cases when we used the evaluation values after alpha-beta with lookahead 6 as the training data. Table 4 shows the case when we use Zebra's evaluation function.

In almost all cases the PUCT is better than the single use of the evaluation function. This was also true for the evaluation functions generated using the 2 and 4 lookahead. There are only two cases were this was not true. In both of them, the PUCT fails to improve upon the result of the single use evaluation function only for the lesser number of simulations. For one of the cases, it is interesting to note that, while the error rate does not improve, the correct move ratio does by 11%.

Figures 2, 3 and 4 illustrate the information contained in Tables 2 and 3. With these figures it becomes clearer how the number of simulations affects the error rate. Interestingly, even though the higher playout number generally performs better, the progress trend seems less obvious for the T-30 and T-40 sets. This suggests that the number of simulations have a bigger impact in early stages of the game. It is however not supported when comparing the average improvement of the error rate from 1000 to 128000 simulations. The average improvements are of 0.339, 0.346 and

Table 2 Experiment results for test data T-20, T-30, and T-40 with the networks trained by training dataset with lookahead 0. Each column shows the error rate and best-move ration in brackets.

Model	Parameter Set	Dataset T-20			Dataset T-30			Dataset T-40		
		single use	1×10^3 simulations	128×10^3 simulations	single use	1×10^3 simulations	128×10^3 simulations	single use	1×10^3 simulations	128×10^3 simulations
C_2F2	Large	1.345 [56]	1.032 [68]	1.149 [67]	3.049 [40]	1.363 [55]	1.221 [58]	4.08 [48]	1.70 [64]	1.34 [68]
	Medium	1.780 [48]	1.221 [66]	0.921 [67]	3.721 [41]	1.372 [62]	1.248 [62]	5.32 [36]	3.34 [54]	2.72 [58]
	Small	2.367 [47]	1.681 [61]	1.273 [65]	4.672 [34]	1.784 [52]	1.680 [53]	5.44 [32]	3.54 [55]	3.36 [55]
C_4F3	Large	1.456 [55]	1.032 [71]	0.815 [72]	3.717 [42]	1.314 [62]	1.240 [61]	3.88 [42]	1.90 [60]	1.82 [59]
	Medium	1.848 [54]	1.568 [65]	1.342 [66]	4.528 [38]	1.830 [54]	1.402 [59]	6.06 [35]	2.56 [54]	2.50 [54]
	Small	2.782 [47]	2.543 [61]	1.917 [61]	6.593 [33]	3.095 [45]	2.326 [49]	6.30 [35]	3.42 [58]	2.68 [57]
$C_2 8F3$	Large	2.352 [42]	0.959 [69]	0.960 [66]	3.358 [33]	1.391 [55]	1.303 [58]	4.60 [43]	1.60 [63]	1.56 [63]
	Medium	2.107 [53]	1.374 [66]	1.226 [68]	4.413 [38]	1.795 [51]	1.566 [56]	5.16 [37]	2.38 [53]	1.78 [60]
	Small	4.329 [41]	2.799 [54]	2.407 [55]	6.088 [35]	2.852 [51]	2.463 [55]	5.78 [38]	3.38 [56]	3.10 [59]
$C_3 2F2$	Large	2.104 [52]	0.994 [72]	0.951 [68]	3.818 [35]	1.517 [55]	1.457 [57]	4.62 [40]	2.10 [62]	1.98 [59]
	Medium	2.208 [49]	1.876 [58]	1.336 [61]	5.865 [30]	1.948 [54]	1.359 [58]	6.38 [37]	3.44 [52]	3.20 [53]
	Small	1.936 [53]	1.847 [62]	1.422 [62]	5.959 [31]	2.161 [50]	1.542 [55]	6.26 [34]	3.34 [51]	3.62 [48]
$C_3 4F3$	Large	2.244 [50]	1.087 [66]	1.066 [66]	4.642 [37]	1.562 [55]	1.525 [57]	4.88 [39]	2.16 [59]	2.18 [59]
	Medium	1.771 [48]	2.050 [59]	1.268 [61]	4.255 [38]	1.636 [51]	1.393 [54]	6.40 [34]	1.86 [61]	1.48 [65]
	Small	2.997 [47]	2.501 [57]	2.053 [57]	5.579 [35]	3.546 [45]	2.940 [49]	7.56 [30]	4.06 [45]	3.16 [51]

Table 3 Experiment results for test data T-20, T-30, and T-40 with the networks trained by training dataset with lookahead 6. Each column shows the error rate and best-move ration in brackets.

Model	Parameter Set	Dataset T-20			Dataset T-30			Dataset T-40		
		single use	1×10^3 simulations	128×10^3 simulations	single use	1×10^3 simulations	128×10^3 simulations	single use	1×10^3 simulations	128×10^3 simulations
C_2F2	Large	2.632 [43]	1.920 [56]	1.532 [62]	3.825 [39]	1.606 [51]	1.277 [57]	5.90 [37]	2.68 [51]	2.32 [57]
	Medium	2.673 [41]	1.686 [60]	1.485 [61]	4.901 [36]	3.018 [44]	2.937 [43]	6.42 [33]	3.82 [48]	3.58 [50]
	Small	6.038 [30]	2.643 [52]	1.716 [54]	8.236 [24]	3.334 [40]	3.234 [39]	6.98 [30]	3.56 [54]	2.88 [56]
C_4F3	Large	1.605 [56]	0.781 [67]	0.672 [66]	3.423 [39]	1.604 [51]	1.458 [50]	4.32 [44]	2.96 [50]	1.80 [62]
	Medium	2.328 [51]	1.963 [57]	1.667 [61]	4.121 [40]	2.030 [51]	1.667 [49]	5.46 [36]	2.46 [61]	1.92 [58]
	Small	3.385 [37]	2.198 [55]	2.148 [53]	5.987 [29]	3.288 [41]	2.767 [44]	6.46 [33]	4.84 [39]	3.62 [45]
$C_2 8F3$	Large	1.882 [52]	1.275 [66]	1.201 [66]	5.029 [32]	1.886 [52]	1.520 [52]	4.58 [37]	2.48 [53]	2.16 [49]
	Medium	2.663 [47]	2.056 [62]	1.766 [63]	4.835 [40]	1.883 [53]	1.558 [53]	6.58 [34]	2.88 [51]	3.32 [47]
	Small	4.957 [39]	3.204 [50]	2.441 [57]	6.572 [24]	3.506 [44]	2.786 [45]	7.04 [32]	3.28 [54]	3.32 [51]
$C_3 2F2$	Large	1.970 [51]	1.710 [61]	1.468 [60]	4.706 [38]	1.601 [54]	1.251 [54]	5.84 [34]	2.16 [57]	1.90 [58]
	Medium	2.994 [50]	2.048 [54]	1.704 [59]	6.900 [23]	2.456 [44]	2.078 [50]	6.16 [33]	3.42 [52]	2.86 [55]
	Small	3.570 [42]	3.644 [41]	3.146 [43]	6.118 [32]	3.497 [36]	3.558 [34]	8.20 [30]	5.12 [42]	4.54 [43]
$C_3 4F3$	Large	1.612 [53]	1.459 [65]	0.971 [70]	4.168 [39]	1.653 [57]	1.641 [58]	4.74 [37]	2.46 [53]	2.74 [50]
	Medium	2.588 [47]	2.165 [56]	1.551 [64]	5.470 [31]	2.317 [45]	1.691 [51]	4.84 [42]	2.70 [56]	1.86 [63]
	Small	6.515 [37]	4.378 [46]	3.093 [51]	7.214 [33]	3.722 [40]	2.745 [44]	6.54 [35]	3.58 [45]	3.30 [48]

Table 4 Experiment results for test data T-20, T-30, and T-40 with the evaluation function of Zebra. Each column shows the error rate and best-move ration in brackets.

	Dataset T-20			Dataset T-30			Dataset T-40		
	single use	1×10^3 simulations	128×10^3 simulations	single use	1×10^3 simulations	128×10^3 simulations	single use	1×10^3 simulations	128×10^3 simulations
Pure Zebra	2.098 [49]	0.665 [70]	0.645 [69]	3.238 [36]	1.049 [61]	1.058 [61]	4.68 [41]	1.56 [59]	1.42 [62]

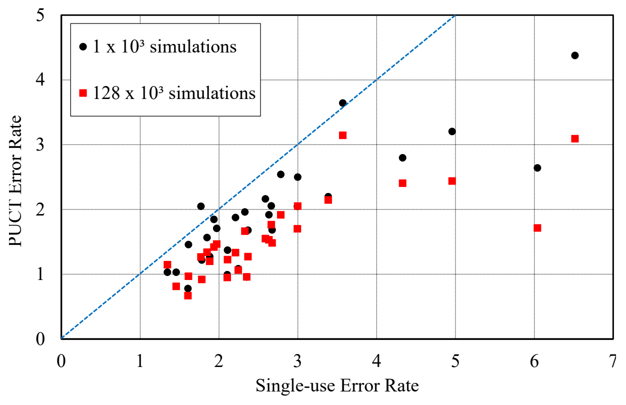


Fig. 2 Comparing error rates of PUCT algorithm with respect to those of single-use evaluation function for dataset T-20

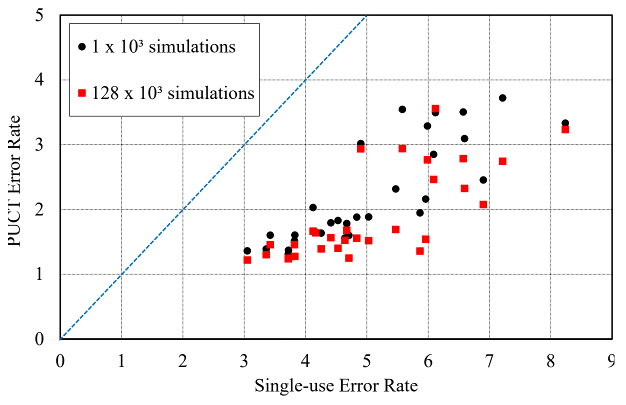


Fig. 3 Comparing error rates of PUCT algorithm with respect to those of single-use evaluation function for dataset T-30

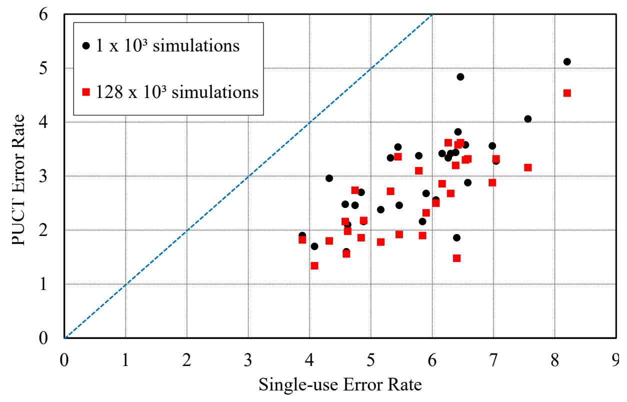


Fig. 4 Comparing error rates of PUCT algorithm with respect to those of single-use evaluation function for dataset T-40

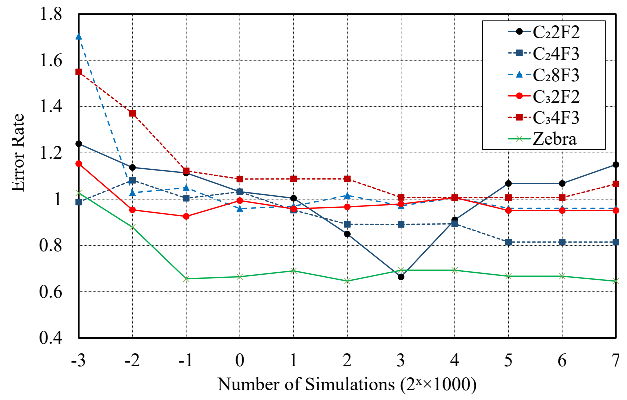


Fig. 5 PUCT performance evolution of large neural-network evaluation functions trained with lookahead 0

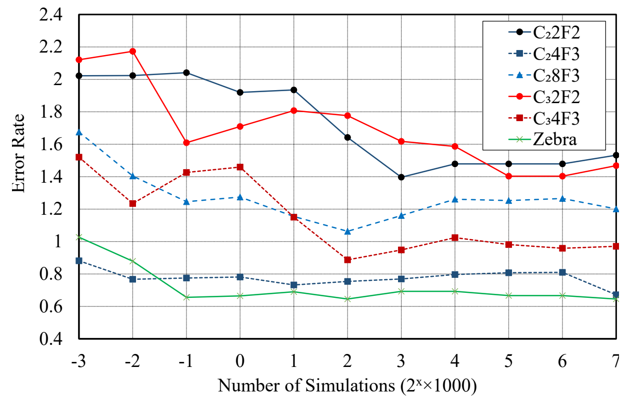


Fig. 6 PUCT performance evolution of large neural-network evaluation functions trained with lookahead 6

0.47 for T-20, T-30 and T-40, respectively. The standard deviation of the error rate values being close to 0.35 for all three sets, the evidence for the visual trend in the graphs is not strong enough. Arguably the number of simulations might even have a bigger impact in the end game than the early game. Note that many simulations went down to the end game with the larger number of simulations in our experiments.

4.2 Influence of the Number of Simulations

Looking at figures 5 and 6, there are mostly variations in error rate up to 16000 simulations and a more stable result afterwards. Moreover, the error rate of the single evaluation function is linked to the value around which the error rate stabilizes as well as the number of simulations before it stabilizes. This means that, the better the single-use error rate is, the faster the PUCT error rate will stabilize and the lower the stable PUCT error rate will be.

Looking at the values obtained for each individual model, these trends still apply across different parameter sets and different training data.

This is something that had already been witnessed in our previous paper [12] which used variations of Zebra as evaluation functions. Having no visible difference between these results and previous ones indicates that the PUCT algorithm probably does not have major differences in behavior depending on the evaluation function being an n-tuple network or a neural network. In other words, according to our data, the evaluation function being linear or non-linear does not seem to have an impact on PUCT.

4.3 Evaluation function performance

We examined the performance of each of our evaluation functions regarding the replication of Zebra's evaluation values with the same lookahead as their training set. We then compared this performance to the error rate they achieved in our T-20 set to see how the two were related.

To evaluate the replication performance of our evaluation functions, we used sets of 100 positions whose possible moves had not been included in the training data. We evaluated all the possible moves with alpha-beta lookahead 0 to 6 using Zebra's evaluation function. We compared the values from a neural network and those from alpha-beta search (with the same lookahead).

We calculated the absolute difference between the neural network evaluation and the Zebra evaluation it was trained to replicate. Taking the Mean Absolute error over all moves of the 100 positions, we obtained one value per neural network evaluation function representing how well it achieved the task it had been trained for.

Comparing these values to the single use of the evaluation function and the PUCT on the T-20 data set, shows Pearson correlation values of 0.639 between mean absolute error and single-use evaluation function error rate. The correlation between mean absolute error and PUCT error rate is decomposed by number of simulations and illustrated in Table 5. They range from 0.561 to 0.698. Such positive correlations were expected since the Zebra function used to create the training data for the neural-network evaluation functions is also used to evaluate the boards of the T-20 set albeit with a different lookahead making the T-20 evaluations much more precise.

The correlation values become better with increasing number of simulations. This is most likely due to the PUCT algorithm compensating for the neural-network evaluation function being trained using evaluations with a smaller lookahead than those of the T-20 set. However, the correlation of mean absolute error and single use of evaluation function is higher than that of mean absolute error and PUCT until 2000 simulations. We know from data shown above that this smaller correlation is not reflected in PUCT performance being worse than the single use of evaluation function at lower simulations numbers. This might point to expansion of the PUCT tree having a bigger impact in performance than the quality of the evaluation function in the early stages of

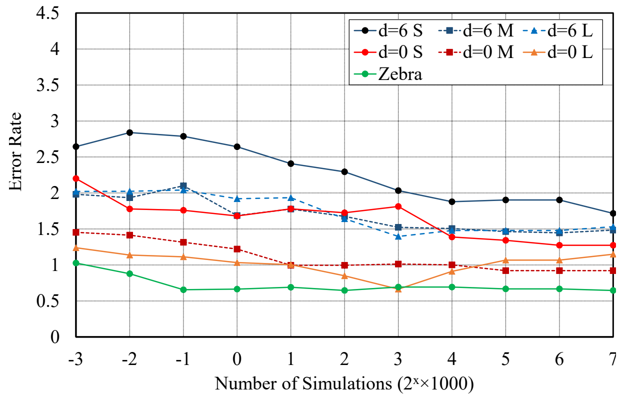


Fig. 7 Evolution of PUCT error rate in model C_22F2 for all parameter sets and training data lookahead of 0 and 6

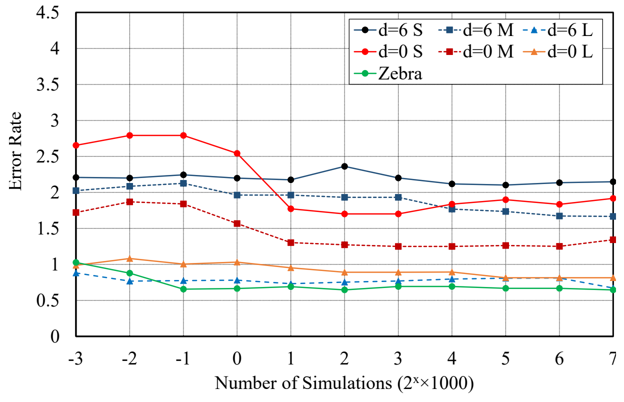


Fig. 8 Evolution of PUCT error rate in model C_24F3 for all parameter sets and training data lookahead of 0 and 6

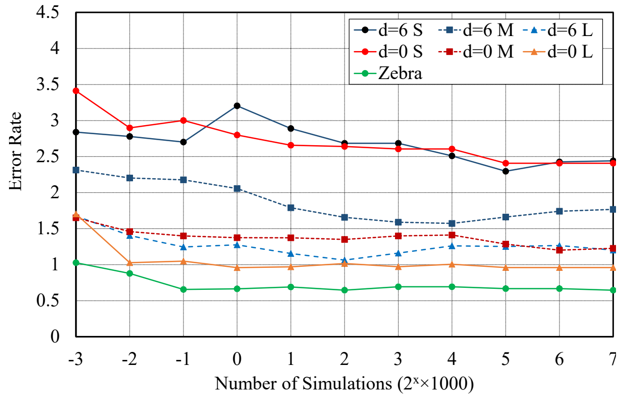


Fig. 9 Evolution of PUCT error rate in model C_28F3 for all parameter sets and training data lookahead of 0 and 6

Table 5 Correlation of all PUCT players error rate with the mean absolute error

playouts	Correlation
125	0.561
250	0.599
500	0.583
1000	0.604
2000	0.639
4000	0.644
8000	0.652
16000	0.649
32000	0.678
64000	0.698
128000	0.678

simulations.

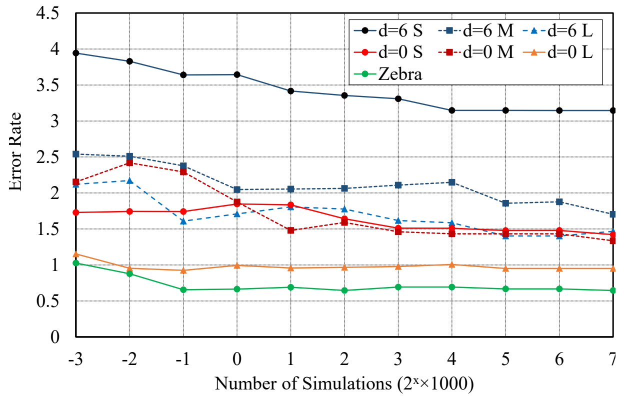


Fig. 10 Evolution of PUCT error rate in model C_32F2 for all parameter sets and training data lookahead of 0 and 6

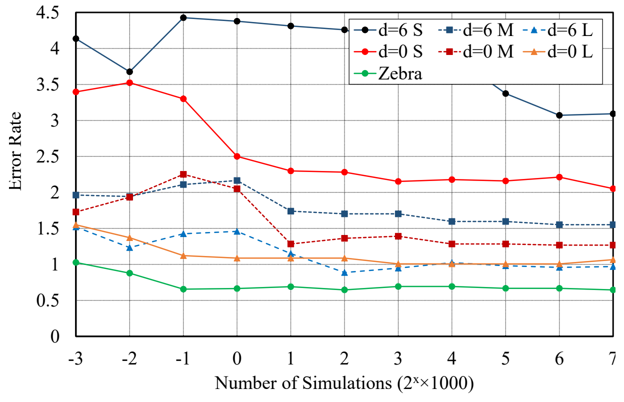


Fig. 11 Evolution of PUCT error rate in model C_34F3 for all parameter sets and training data lookahead of 0 and 6

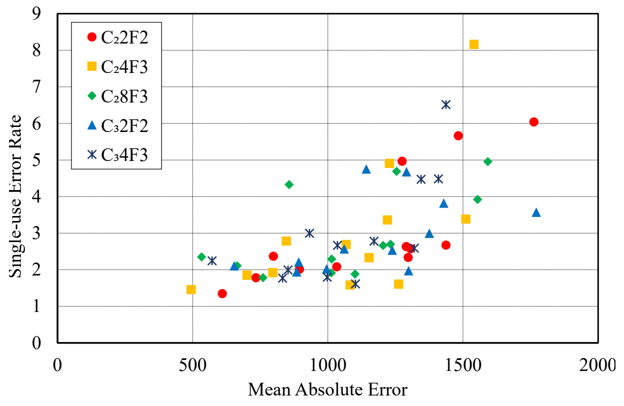


Fig. 12 Single-use error rate in relation to mean absolute error of neural-network evaluation function

4.4 AlphaBeta search comparison

For the final part of our experimentation, we wished to compare PUCT to Alpha-Beta move selection. To do so, we ran it with all neural network evaluation functions on T-20, T-30 and T-40 using a lookahead of 0 to 6 (0 is equivalent to single-use evaluation function). The results using the smallest and largest lookahead are presented in Tables 6 and 7. Comparing these tables to the previous ones showing results of similar experimentation using PUCT (Tables 2 and 3), we can see that Alpha-Beta with lookahead 6 outperforms PUCT using 128000 simulations with regards to error rate and correct move ratio. The average improvement in error rate for the T-20 set from Alpha-Beta 0 to 6 is 1.644 while it is only 1.364 from single-use evaluation function

Table 6 Results of AlphaBeta search using neural network evaluation functions trained with training data with depth 0. Each column shows the error rate and best-move ratio in brackets.

	Parameter	Dataset T-20			Dataset T-30			Dataset T-40		
Model	Set	Single use	Lookahead 2	Lookahead 6	Single use	Lookahead 2	Lookahead 6	Single use	Lookahead 2	Lookahead 6
	Large	1.345 [56]	1.417 [62]	0.634 [73]	3.048 [40]	2.104 [46]	0.924 [62]	4.08 [48]	2.58 [57]	1.90 [58]
C_2F2	Medium	1.780 [48]	1.493 [59]	0.649 [68]	3.721 [41]	1.987 [46]	1.099 [64]	5.32 [36]	3.78 [51]	2.16 [52]
	Small	2.367 [47]	1.881 [57]	1.091 [66]	4.672 [34]	2.803 [43]	1.284 [60]	5.44 [32]	4.22 [48]	2.46 [52]
	Large	1.456 [55]	1.210 [67]	0.671 [74]	3.716 [42]	1.977 [48]	0.914 [64]	3.88 [42]	3.16 [49]	1.74 [59]
C_24F3	Medium	1.847 [54]	1.768 [58]	0.757 [72]	4.528 [38]	2.583 [42]	1.419 [57]	6.06 [35]	4.06 [45]	2.32 [47]
	Small	2.782 [47]	2.520 [51]	1.202 [65]	6.592 [33]	4.247 [40]	2.293 [48]	6.30 [35]	4.66 [46]	3.70 [48]
	Large	2.351 [42]	1.308 [60]	0.662 [72]	3.358 [33]	1.646 [50]	1.108 [62]	4.60 [43]	2.28 [56]	1.42 [68]
C_28F3	Medium	2.107 [53]	2.018 [59]	0.629 [74]	4.412 [38]	2.512 [43]	1.286 [61]	5.16 [37]	3.78 [42]	1.98 [55]
	Small	4.328 [41]	2.838 [50]	2.300 [57]	6.088 [35]	4.129 [42]	2.357 [51]	5.78 [38]	5.52 [37]	3.42 [46]
	Large	2.103 [52]	1.896 [57]	0.807 [70]	3.818 [35]	2.115 [47]	1.018 [63]	4.62 [40]	3.56 [52]	1.94 [55]
C_32F2	Medium	2.208 [49]	2.810 [50]	1.459 [62]	5.865 [30]	2.505 [47]	1.628 [51]	6.38 [37]	4.20 [46]	2.36 [53]
	Small	1.935 [53]	2.435 [59]	1.598 [63]	5.959 [31]	3.111 [44]	1.265 [56]	6.26 [34]	4.56 [39]	2.30 [52]
	Large	2.243 [50]	1.728 [56]	0.892 [65]	4.641 [37]	1.904 [49]	0.827 [66]	4.88 [39]	2.78 [56]	1.46 [63]
C_34F3	Medium	1.770 [48]	1.802 [57]	1.004 [72]	4.255 [38]	2.514 [46]	1.259 [59]	6.40 [34]	3.74 [50]	1.74 [60]
	Small	2.997 [47]	2.419 [53]	1.938 [55]	5.579 [35]	4.247 [42]	2.471 [44]	7.56 [30]	4.52 [48]	2.92 [49]

Table 7 Results of AlphaBeta search using neural network evaluation functions trained with training data with depth 6. Each column shows the error rate and best-move ratio in brackets.

	Parameter	Dataset T-20			Dataset T-30			Dataset T-40		
Model	Set	Single use	Lookahead 2	Lookahead 6	Single use	Lookahead 2	Lookahead 6	Single use	Lookahead 2	Lookahead 6
	Large	2.632 [43]	2.291 [49]	1.343 [58]	3.824 [39]	2.227 [44]	1.529 [57]	5.90 [37]	4.10 [43]	2.36 [53]
C_2F2	Medium	2.672 [41]	2.072 [54]	1.082 [63]	4.900 [36]	3.480 [40]	1.578 [52]	6.42 [33]	4.78 [45]	3.70 [45]
	Small	6.038 [30]	4.331 [38]	2.663 [49]	8.236 [24]	4.947 [35]	2.269 [50]	6.98 [30]	5.94 [31]	2.56 [47]
	Large	1.604 [56]	1.529 [57]	0.819 [68]	3.422 [39]	2.046 [48]	1.004 [62]	4.32 [44]	3.84 [43]	2.06 [56]
C_24F3	Medium	2.328 [51]	1.930 [55]	0.945 [70]	4.121 [40]	2.307 [50]	1.123 [60]	5.46 [36]	3.94 [45]	2.32 [53]
	Small	3.385 [37]	3.228 [46]	1.213 [63]	5.987 [29]	4.410 [32]	2.589 [45]	6.46 [33]	4.30 [37]	2.18 [54]
	Large	1.881 [52]	1.614 [60]	0.961 [68]	5.028 [32]	2.008 [49]	1.064 [61]	4.58 [37]	3.58 [44]	1.72 [59]
C_28F3	Medium	2.663 [47]	2.937 [51]	1.162 [66]	4.834 [40]	2.869 [46]	1.444 [57]	6.58 [34]	4.20 [40]	2.12 [54]
	Small	4.956 [39]	3.614 [47]	2.248 [55]	6.571 [24]	3.684 [35]	2.223 [48]	7.04 [32]	5.12 [37]	2.08 [61]
	Large	1.970 [51]	2.706 [53]	0.921 [72]	4.706 [38]	2.838 [45]	1.359 [58]	5.84 [34]	4.06 [47]	2.10 [58]
C_32F2	Medium	2.993 [50]	2.837 [44]	1.343 [58]	6.899 [23]	3.368 [37]	2.259 [52]	6.16 [33]	3.72 [46]	2.18 [58]
	Small	3.569 [42]	3.621 [42]	1.916 [47]	6.117 [32]	4.183 [38]	2.404 [43]	8.20 [30]	6.00 [31]	3.54 [45]
	Large	1.612 [53]	1.951 [63]	0.900 [66]	4.167 [39]	1.712 [58]	1.009 [64]	4.74 [37]	3.62 [42]	1.70 [63]
C_34F3	Medium	2.588 [47]	2.339 [54]	1.513 [61]	5.470 [31]	2.984 [43]	1.905 [54]	4.84 [42]	4.06 [45]	2.48 [56]
	Small	6.514 [37]	3.988 [42]	2.216 [57]	7.214 [33]	5.823 [31]	2.646 [48]	6.54 [35]	5.24 [33]	3.08 [53]

to PUCT with 128'000 simulations. Similar trends were seen for T-30 and T-40.

5. Related Work

The study most related to this work is the analysis of the AlphaZero algorithm by Nakayashiki and Kaneko [14]. They used a small (but still nontrivial) game called Dobutsu Shogi, a small variant of Japanese chess, which was strongly solved by Takana [22]. In the study, they focused on the quality of reinforcement learning in the AlphaZero approach. Since win/draw/lose is known for each game position, they can evaluate the results quantitatively and accurately. They also discussed the effects of PUCT algorithm by using a single pair of evaluation functions and by changing a hyperparameter C for exploration. The experiment results suggested that the value $C = 1$ would be better for a large number of simulations than that used in AlphaZero.

Takeuchi [20] studied another aspect of MCTS (and PUCT) algorithms focusing on the effects of neural-network evaluation functions. He focused on the changes of selected moves when the number of playouts were increased. The experiment results suggested that the use of neural-network evaluation function showed a gap for even fundamental measures such as winning rate and number of playouts for the best moves. It was also reported that the MCTS algorithm did not show symmetric effects of evaluation functions that the alpha-beta search did.

This study focused on a different aspect of the PUCT algorithms, i.e., the effects of nonlinear neural-network evaluation functions with different quality.

6. Conclusion

In this work we have tried to analyze the effects of the evaluation function's influence on AlphaGo's PUCT algorithm using Othello as a testing environment. Our previous work having focused on the same question using linear evaluation functions, we decided to examine non-linear functions in this paper by creating several neural networks with different structures trained to mimic Zebra evaluations with different lookaheads. We then tested the player using sets of test positions with very precise scores.

We found that PUCT always improves on the performance of the single use of evaluation functions no matter how accurate they were. Moreover, comparing the results in this paper to those presented in our previous one, we observed very similar trends between the use of n-tuple network and neural network evaluation functions. We thus concluded that there is likely no difference in PUCT behavior resulting from the use of linear or non-linear evaluations functions.

While we did extend the number of testing board sets from our previous study, there are still large periods of the games we have not analyzed and witnessing evolution of error rate through the game might bring interesting insights. The same consideration could be brought to the evolving number of simulations; while we did check progress at many numbers of simulations, it might be interesting to plot error rates after each new simulation. Finally, instead of using many different evaluation functions to analyze, we could select only 2 or 3 of them (linear and non-linear) and design them to have extremely similar performances for certain measures prior to using them in PUCT to have a deeper look at how similar or different their results become post PUCT.

References

[1] Andersson, G.: Zebra, `http://radagast.se/othello/zebra.html` (2011).

[2] Browne, C. B., Whitehouse, D., Cowling, P. I. and Samothrakis, S.: A Survey of Monte Carlo Tree Search Methods, *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, No. 1, pp. 1–43 (2012).

[3] Chaslot, G., Fiter, C., Hoock, J.-B., Rimmel, A. and Teytaud, O.: Adding Expert Knowledge and Exploration in Monte-Carlo Tree Search, *Proceedings of 12th International Conference on Advances in Computer Games* (*ACG'09*), pp. 1–13 (2009).

[4] Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search, *Proceedings of 5th International Conference on Computers and Games* (*CG 2006*), pp. 72–83 (2006).

[5] Gelly, S. and Silver, D.: Combining Online and Offline Knowledge in UCT, *Proceedings of 24th International Conference on Machine Learning* (*ICML'07*), pp. 273–280 (2007).

[6] Gelly, S. and Silver, D.: Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go, *Artificial Intelligence*, Vol. 175, pp. 1856–1875 (2011).

[7] Kloetzer, J., Iida, H. and Bouzy, B.: The Monte-Carlo Approach in Amazons, *Proceedings of Computer Games Workshop*, pp. 113–124 (2007).

[8] Kocsis, L. and Szepesvári, C.: Bandit Based Monte-Carlo Planning, *Proceedings of 17th European Conference on Machine Learning* (*ECML 2006*), pp. 282–293 (2006).

[9] Lorentz, R.: Amazons Discover Monte-Carlo, *Proceedings of 6th International Conference on Computers and Games* (*CG 2008*), pp. 13–24 (2008).

[10] Lorentz, R.: Using Evaluation Functions in Monte-Carlo Tree Search, *Theoretical Computer Science*, Vol. 644, pp. 106–113 (2016).

[11] Lorentz, R. and Horey, T.: Programming Breakthrough, *Proceedings of 8th International Conference on Computers and Games* (*CG 2013*), pp. 49–59 (2013).

[12] Matsuzaki, K.: Empirical Analysis of PUCT Algorithm with Evaluation Functions of Different Quality, *Conference on Technologies and Applications of Artificial Intelligence* (*TAAI 2018*), IEEE Computer Society, pp. 142–147 (2018).

[13] Matsuzaki, K. and Kitamura, N.: Do Evaluation Functions Really Improve Monte-Carlo Tree Search? — Empirical Analysis Using Othello, *Proceedings of the 10th International Conference on Computers and Games* (*CG2018*) (2018).

[14] Nakayashiki, T. and Kaneko, T.: A Survey on AlphaZero Algorithm through Dobutsu Shogi, *Proceedings of the 24th Game Programming Workshop 2019*, pp. 86–93 (2019). in Japanese.

[15] Rosin, C. D.: Multi-armed Bandits with Episode Context, *Annals of Mathematics and Artificial Intelligence*, Vol. 61, No. 3, pp. 203–230 (2011).

[16] Sahara, K. and Matsuzaki, K.: Effects of Noise Addition to Evaluation Functions in Monte-Carlo Tree Search, IPSJ SIG Technical Reports Vol. 2020-GI-43 No. 24 (2020). in Japanese.

[17] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D.: Mastering the Game of Go with Deep Neural Networks and Tree Search, *Nature*, Vol. 529, No. 7587, pp. 484–489 (2016).

[18] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. and Hassabis, D.: A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-play, *Science*, Vol. 362, No. 6419, pp. 1140–1144 (2018).

[19] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. and Hassabis, D.: Mastering the Game of Go without Human Knowledge, *Nature*, Vol. 550, pp. 354–359 (2017).

[20] Takeuchi, S.: Behavior of Monte-Carlo Tree Search with Increase of Number of Playouts, *24th Game Programming Workshop*, pp. 66–72 (2019). in Japanese.

[21] Takeuchi, S. and Kaneko, T.: Estimating Ratings of Computer Players by the Evaluation Scores and Principal Variations in Shogi, *ACIT-CSI'15 Proceedings of the 2015 3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence* (2015).

[22] Tanaka, T.: An Analysis of a Board Game "Dobutsu Shogi", IPSJ SIG Technical Reports Vol.2009-GI-22 (2009). in Japanese.

[23] Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D. and Silver, D.: AlphaStar: Mastering the Real-Time Strategy Game StarCraft II, `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/` (2019).