IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 December 2022, accepted 30 January 2023, date of publication 21 February 2023, date of current version 24 February 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3246638*

RESEARCH ARTICLE

# Analyses of Tabular AlphaZero on Strongly-Solved Stochastic Games

**CHU-HSUAN HSUEH[1], (Member, IEEE), KOKOLO IKEDA[1], I-CHEN WU[2], (Senior Member, IEEE), JR-CHANG CHEN[3], AND TSAN-SHENG HSU[4], (Senior Member, IEEE)**

[1]School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923-1211, Japan
[2]Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan
[3]Department of Computer Science and Information Engineering, National Taipei University, New Taipei City 23741, Taiwan
[4]Institute of Information Science, Academia Sinica, Taipei 11529, Taiwan

Corresponding author: Chu-Hsuan Hsueh (hsuehch@jaist.ac.jp)

This work was supported in part by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 20K19946; and in part by the National Science and Technology Council (NSTC), Taiwan, through Pervasive Artificial Intelligence Research (PAIR) Labs, under Grant 110-2221-E-A49-067-MY3 and Grant 111-2221-E-305-006-MY2.

**ABSTRACT** The AlphaZero algorithm achieved superhuman levels of play in chess, shogi, and Go by learning without domain-specific knowledge except for game rules. This paper targets stochastic games and investigates whether AlphaZero can learn theoretical values and optimal play. Since the theoretical values of stochastic games are expected win rates, not a simple win, loss, or draw, it is worth investigating the ability of AlphaZero to approximate expected win rates of positions. This paper also thoroughly studies how AlphaZero is influenced by hyper-parameters and some implementation details. The analyses are mainly based on AlphaZero learning with lookup tables. Deep neural networks (DNNs) like the ones in the original AlphaZero are also experimented and compared. The tested stochastic games include reduced and strongly-solved variants of Chinese dark chess and EinStein würfelt nicht!. The experiments showed that AlphaZero could learn policies that play almost optimally against the optimal player and could learn values accurately. In more detail, such good results were achieved by different hyper-parameter settings in a wide range, though it was observed that games on larger scales tended to have a little narrower range of proper hyper-parameters. In addition, the results of learning with DNNs were similar to lookup tables.

**INDEX TERMS** AlphaZero, board games, Chinese dark chess, EinStein würfelt nicht!, reinforcement learning, stochastic games, tabular.

## I. INTRODUCTION

In 2017, Silver et al. [1] presented a program named AlphaGo Zero, which achieved a superhuman level in the game of Go. Starting from random play, the program was trained from self-play games and did not use domain-specific knowledge other than game rules. They further generalized the approach to the AlphaZero algorithm and successfully applied it to chess and shogi as well as Go [2]. The trained programs defeated world-champion programs in each of the three games, which represents a milestone in the field of artificial intelligence.

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Alawneh.

The AlphaZero algorithm employed neural networks with two output heads, value and policy, to evaluate board positions and select actions, respectively. The neural networks were combined into a variant of Monte-Carlo tree search (MCTS) for both generating self-play games to learn and playing games. The algorithm contained several hyper-parameters related to MCTS that might influence the learning in different aspects, such as speed, stability, and the quality of the learned policies and values. According to Silver et al. [2], most of the hyper-parameters followed those tuned in AlphaGo Zero [1] by Bayesian optimization [3] with some exceptions. It is interesting to investigate how and how much hyper-parameters influence the algorithm.

Moreover, it is also worth studying whether the AlphaZero algorithm learns theoretical values and optimal play even

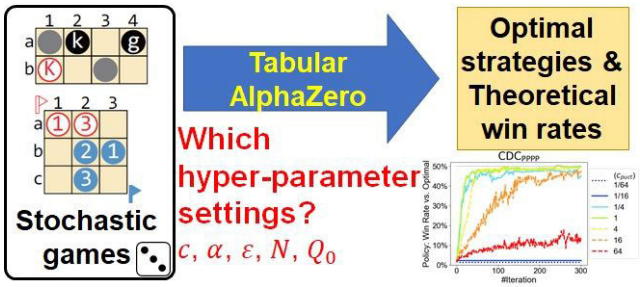



**FIGURE 1.** An overview of the analyses of tabular AlphaZero on strongly-solved stochastic games.

in stochastic games. The algorithm has been demonstrated to achieve world-champion levels in deterministic games, including chess, shogi, and Go. The theoretical values of the positions in such games are expected to be a win, loss, or draw. In contrast, in stochastic games such as Chinese dark chess (abbr. CDC) [4] and EinStein würfelt nicht! (abbr. EWN) [5], the theoretical values are expected win rates. It is interesting to study the ability of the AlphaZero algorithm to approximate expected win rates.

In this paper, reduced and strongly-solved game variants [6], [7] are targeted instead of the standard versions for two main reasons. The first and most important one is that the learned values and policies can be directly compared with the theoretical values and the optimal play. By doing so, the difference between the AlphaZero algorithm and the optimal strategy can be analyzed. Second, learning games on smaller scales requires fewer computation resources. Different hyper-parameter settings can be thoroughly investigated, even for high numbers of simulations in MCTS, which is infeasible in games on larger scales. A setting can also be tried several times to verify the repeatability.

As the first step, lookup tables keep for each position the policy (the probabilities of selecting actions) and the estimated value, instead of using neural networks as function approximators. In this way, it is easier to analyze how and why the learning succeeds or fails since each position in the lookup table is independent. The AlphaZero algorithm employing lookup tables is referred to as the tabular AlphaZero. Lookup tables are feasible since the state spaces of the reduced and strongly-solved game variants are small enough. In addition, neural networks like those used by Silver et al. [2] are employed to see whether the results match lookup tables'.

In the experiments, the tabular AlphaZero is applied to learn strongly-solved variants of CDC [6] and EWN [7]. For both CDC and EWN, several variants differing in board sizes or piece numbers are selected, aiming to provide insights when generalizing the analyses to games on larger scales. The state spaces of these game variants are around the order of $10^5$ or $10^6$. Several hyper-parameters of MCTS for generating self-play games are thoroughly investigated. The algorithm plays almost optimally against the optimal player in many tested settings. Fig. 1 shows an overview of the analyses. In addition, the results of the AlphaZero algorithm with neural networks are similar to those of tabular AlphaZero, suggesting that the analyses through lookup tables are worth referencing in general.

This paper is extended from a preliminary version [8] by investigating more stochastic games, designing new metrics for evaluating policies and values learned by the tabular AlphaZero (policy loss and mean absolute error in Section IV), analyzing the tabular AlphaZero more thoroughly (Sections V to VII), and making comparisons between learning using lookup tables and neural networks (Section VIII).

The rest of the paper is structured as follows. Section II describes background knowledge, including the AlphaZero algorithm and the games of CDC and EWN. Section III then introduces the tabular AlphaZero applied to the strongly-solved variants. Section IV proposes metrics for measuring the performance of lookup tables. Section V to VII then present the experiments on MCTS hyper-parameters. Section VIII shows the results of learning with neural networks. Finally, Section IX makes concluding remarks and discusses future research directions.

## II. BACKGROUND

First, Subsection II-A reviews the AlphaZero algorithm. Subsections II-B and II-C then introduce the games of CDC and EWN, along with the strongly-solved variants.

### A. THE ALPHAZERO ALGORITHM

In the AlphaZero algorithm [2], deep neural networks (DNNs) were trained by reinforcement learning to get good policies (i.e., the probability distribution of selecting actions) and predict positions' values (i.e., expected win rates) accurately. The key idea was to make the program become its own teacher through tree search and self-play. The policies and the values generated by DNNs were incorporated into tree search to obtain better policies. The DNNs were then trained to predict the policies after tree search and the final winners in the self-play games. In more detail, the learning algorithm contained two parts, self-play and optimization, executed asynchronously in parallel in a large-scale computation system.

Self-play games were generated by a variant of Monte-Carlo tree search (MCTS) with $N_{sim}$ simulations per turn. MCTS in the AlphaZero algorithm contained three phases in each simulation, which were selection, expansion, and backpropagation. In the selection phase, the PUCT algorithm,

$$\underset{a}{\operatorname{argmax}}\{\frac{W(s,a)}{N(s,a)} + c_{puct} \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}\}, \quad (1)$$

was applied to traverse the search tree to a leaf, where $W(s,a)$ is the total value of action $a$ for node $s$, $N(s,a)$ the visit count, $P(s,a)$ the prior probability of selecting $a$ at $s$, and $c_{puct}$ a coefficient influencing the level of exploration in the search. During the expansion phase, the DNN was used to

evaluate the value $v$ of the leaf node $s_L$ and obtain the policy $\boldsymbol{p}$, the probabilities of legal actions. Without loss of generality, assume that $v$ ranges in $[0, 1]$ and is from the view of the player to play. $s_L$ was then expanded with each of the actions initialized to

$$\begin{aligned} N(s_L, a) &\leftarrow 0, \\ W(s_L, a) &\leftarrow 0, \\ P(s_L, a) &\leftarrow p_a, \end{aligned} \tag{2}$$

where $p_a$ is the probability of selecting action $a$ from the DNN.[1] Backpropagation updated the traversed nodes $s$ from the leaf back to the root by

$$\begin{aligned} N(s, a) &\leftarrow N(s, a) + 1, \\ v &\leftarrow 1 - v, \\ W(s, a) &\leftarrow W(s, a) + v. \end{aligned} \tag{3}$$

Finally, the action to be actually selected at the root was chosen in proportion to the visit counts. At the end of a self-play game, the policies $\boldsymbol{\pi}, \pi_a \propto N(s, a)$, of all board positions $s$ and the final outcome $z$ of the game were saved as training data.

To ensure all legal actions at the root may be tried, Dirichlet noise was added to the prior probabilities at the root node. The prior probability $P(s_0, a)$ was modified to

$$(1 - \varepsilon)p_a + \varepsilon\eta_a, \tag{4}$$

where $\boldsymbol{\eta}$ is sampled from a symmetric Dirichlet distribution $\mathrm{Dir}(\alpha)$ with $\alpha$ controlling the level of concentration, and $\varepsilon$ is the weight of the noise. The distribution $\mathrm{Dir}(\alpha)$ was sampled when the root was expanded. Under the same numbers of samples, lower $\alpha$ generally leads to more concentrated distributions and thus higher levels of exploration. The hyper-parameters related to MCTS in the AlphaZero algorithm generally followed those selected by Bayesian optimization in AlphaGo Zero [1], except the $\alpha$ in the Dirichlet noise and the number of simulations per turn for self-play games.

To optimize policy and value networks, the AlphaZero algorithm randomly sampled positions from some most recent self-play games. The DNNs were then optimized by stochastic gradient descent using the positions and the corresponding policies and values. The resulting programs defeated world-champion ones in the games of chess, shogi, and Go.

As a milestone algorithm that mastered chess, shogi, and Go without game-specific knowledge, several research groups have tried to reimplement the algorithm in various games, e.g., ELF OpenGo [9], Leela Zero [10], Polygames [11], and CLAP [12]. Silver et al.'s implementation was also released in an open-sourced project, OpenSpiel [13], a few years after the paper's publication. Among the projects, ELF OpenGo and Leela Zero were specifically for the game of Go; Polygames, CLAP, and OpenSpiel were general frameworks that could be applied to different board games, including stochastic games.

[1]For newly expanded actions $a$ from leaf node $s_L$, the $W(s_L, a)/N(s_L, a)$ becomes undefined 0/0. Several ways of interpretation will be discussed and experimented in Section VII.

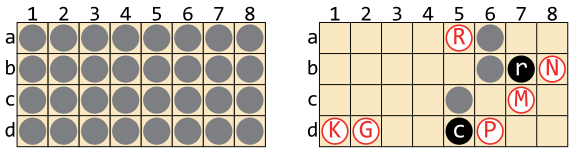



**FIGURE 2.** (a) The initial position and (b) an example of position for 4 × 8 CDC.

One major difference between deterministic and stochastic games is that the search trees of the latter involve chance nodes. As more general extensions of the AlphaZero algorithm, Schrittwieser et al. [14] proposed MuZero, and Antonoglou et al. [15] proposed Stochastic MuZero. However, none of them investigated whether the algorithms could learn the optimal policies and theoretical values nor how hyper-parameters influenced the algorithms. Note that for imperfect information games such as poker that involve another aspect of uncertainty, opponent modeling or mixed strategies [16] may be required. Whether the AlphaZero algorithm is applicable to imperfect information games is still an open problem and is out of this paper's scope.

### B. CHINESE DARK CHESS

Chinese dark chess (CDC) is a two-player zero-sum stochastic board game widely played in Taiwan [4], [17], and also a game played in Computer Olympiad since 2010 [18], [19]. The game is played on a 4 × 8 board with 32 pieces, 16 for each of the two colors, red and black. The 16 pieces are one king (K/k), two guards (G/g), two ministers (M/m), two rooks (R/r), two knights (N/n), two cannons (C/c), and five pawns (P/p). Red pieces are abbreviated by uppercase letters and black pieces by lowercase letters.

At the start of a game, all 32 pieces are placed faced-down, randomly shuffled, and put in the squares on the board, as shown in Fig. 2a. The first player reveals one piece to make it face up and owns the set of pieces with the same color as the revealed one. The second player automatically owns the other set. In other words, the two players' colors are decided after the first action. The two players perform actions in turn, either revealing one of the faced-down pieces or moving one of their own pieces. The information conveyed by revealing or moving pieces can be observed by both players.

The pieces are moved according to the following rules. First, a piece other than cannons can be moved to one of its horizontally or vertically adjacent squares either without pieces or with an opponent's piece that it can capture. The capturing relation is determined by the ranks of the pieces, which from the highest to the lowest are kings, guards, ministers, rooks, knights, cannons, and pawns. In general, a piece can capture the opponent's pieces with equal or lower

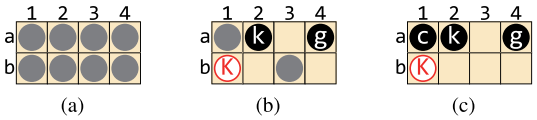



**FIGURE 3.** (a) The initial, (b) an example, and (c) another example of positions for 2 × 4 CDC.

ranks. However, kings cannot capture pawns while pawns can capture kings. For example, the r at b7 in Fig. 2b can be moved to a7 (empty) and b8 (capture) but not b6 and c7. The second rule is for cannons, which move to empty squares as other pieces do but have a special jumping rule. More specifically, cannons can capture all types of opponent's pieces in the same row or the same column with exactly one piece in between. For example, the c at d5 in Fig. 2b can capture the R at a5 or the K at d1.

Two ways for a player to win a game are capturing all the opponent's pieces or making the opponent have no legal actions. A game ends as a draw when both players do not capture or reveal any piece within 40 plies or repeat the same position three times.

#### 1) 2 × 4 CHINESE DARK CHESS

Chang et al. [6], [20] introduced and solved a reduced version of CDC, called 2 × 4 CDC. The board size was reduced to 2 × 4, and only eight pieces were used. The initial position is shown in Fig. 3a. In 2 × 4 CDC, 24 symmetric and non-equivalent material combinations (combinations of pieces) were considered. With symmetric material combinations, both players have the same set of pieces at the start of a game. For example, 'KGGM vs. kggm' is symmetric while 'KGGM vs. kggr' is not. As for equivalence [21], material combinations with the same capturing relations between pieces were only counted once. An example is that 'KGGR vs. kggr' is equivalent to 'KGGM vs. kggm' and thus is not counted.

Chang et al. [6], [20] assigned theoretical values to the positions from the retrograde analysis in an expectiminimax manner. A position was defined by the set of remaining pieces, the configuration of the board, and the player to move. The theoretical values were from the view of the player to move. For a deterministic position, which has at most one piece type faced-down, the theoretical value was one of 1 (win), 0 (draw), and -1 (loss). An example is shown in Fig. 3c, a position from a game of 'KGCP vs. kgcp.' The result is a win for the black player. For a stochastic position, the game outcomes might vary according to different piece types revealed. Thus, the theoretical value was a real number in the range of $[-1, 1]$, where -1 indicated a 100% loss and 1 a 100% win. In this paper, the theoretical values are linearly scaled to the range of $[0, 1]$ without loss of generality. For the example of Fig. 3b from 'KGCP vs. kgcp,' assuming the red player's turn and the unrevealed pieces to be Pc, the theoretical value is 0.250. The red player has a probability of 50% to draw the game at best and loses otherwise.

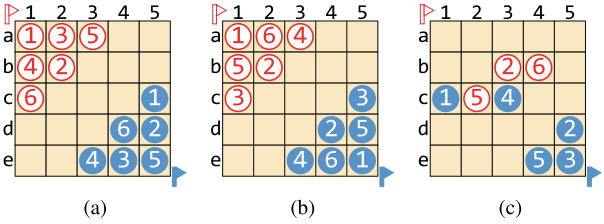



**FIGURE 4.** (a) An initial position, (b) another initial position, and (c) an example of position for EWN.

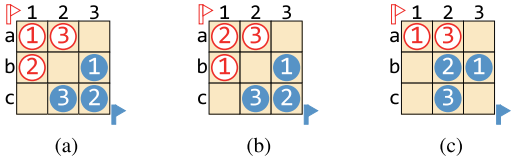



**FIGURE 5.** (a) An initial position, (b) another initial position, and (c) an example of position for 3 × 3 EWN.

### C. EINSTEIN WÜRFELT NICHT!

EinStein würfelt nicht! (EWN) is a two-player zero-sum stochastic board game designed by Althöfer in 2004 [5], and also a game played in Computer Olympiad since 2011 [19]. The game is played with a six-sided dice on a 5 × 5 square board. Two players, colored red and blue, own six pieces numbered 1 to 6, respectively. At the beginning of a game, red and blue pieces are placed in the top-left and bottom-right corners, respectively, as shown in Figs. 4a and 4b. The initial placements of pieces are random or decided by the players. For the case that players simultaneously decide the placements, Nash equilibrium and mixed strategies should be considered, which may not be suitable for the AlphaZero algorithm. For simplicity, this paper only discusses the case of random placements.

The rules do not specify which player to play first. Without loss of generality, the red player moves first in this paper. The two players perform actions in turn, which consist of rolling the dice and then moving the piece of the rolled number. A piece can be moved one square vertically, horizontally, or diagonally toward the opponent's corner. More specifically, red pieces can be moved one square to down, right, or down-right, and the opposite directions for blue pieces. If the destination of a move contains a piece, the piece is captured and removed from the board regardless of the color. For example, the red 2 at b3 in Fig. 4c can capture the red 6 at b4 or the blue 4 at c3. When a player does not have the piece with the rolled number, the player can move a remaining piece with a number either the next-highest or the next-lowest to the rolled number. For example, when the dice number is 3, the red player in Fig. 4c can move either 2 or 5. A player wins by moving one piece to the farthest opponent's corner or capturing all opponent's pieces. In other words, a game always has a winner and never results in draws.

#### 1) STRONGLY-SOLVED EINSTEIN WÜRFELT NICHT!

Bonnet and Viennot [7] solved reduced versions of EWN on different board sizes with up to six pieces by the expectiminimax algorithm. For variants with $k$ initial pieces, $k$-sided dices are used. Figs. 5a and 5b show two initial positions of $3 \times 3$ EWN where each player owns three pieces. The theoretical value, i.e., the expected win rate, of Fig. 5a is 0.700. In more detail, the values for dice numbers 1 to 3 are 0.444, 0.827, and 0.827, respectively. For Fig. 5b, where only red `1` and `2` are swapped compared to Fig. 5a, the theoretical value becomes 0.588, the average of 0.654, 0.457, and 0.654 for dice numbers 1 to 3.

As for positions in EWN, since legal actions are different for different dice numbers, the definition in this paper includes dice numbers in addition to the configuration of the board and the player to move. For example, for the board configuration in Fig. 5c, assuming red's turn, when the dice number is 1, legal actions include a1→b1, a1→a2, and a1→b2. The theoretical value of this position is 0.111 by taking a1→b2; when taking the other two actions, the player loses. When the dice number is 2, since red `2` does not exist, the player can decide to move either red `1` or red `3`. Thus, a total of six legal actions are available, where a2→b2, a2→a3, and a2→b3 are for red `3`. The theoretical value is 0.222 by taking a2→b3, while the expected win rates for a2→b2 and a2→a3 are 0 and 0.049, respectively.

## III. TABULAR ALPHAZERO

As the first step of the investigation, this work replaces the DNNs in the AlphaZero algorithm with lookup tables in order to simplify the analyses, where lookup tables do not perform feature extraction. Since the state spaces of the employed game variants are small enough, lookup tables are feasible. In a lookup table, each position has its own policy and value. The policy is represented by a vector of real numbers $\hat{\boldsymbol{p}}$. Similar to Silver et al.'s design [2], a policy contains all possible actions of the employed action representation, even though some actions may be illegal for the corresponding position. More formally, a policy is a probability distribution for all actions in the set $A = \bigcup_{\{s \in S\}} A(s)$, where $S$ is the state space and $A(s)$ the set of legal actions at position $s$. The representation is redundant for lookup tables; however, it is general and applicable to neural networks. In this paper, $|A|$ is 40 for $2 \times 4$ CDC[2] and 6 for EWN.[3] The probability distribution $\boldsymbol{p}'$ of all actions is calculated by the softmax function [22],

$$p'_a = \frac{e^{\hat{p}_a}}{\sum_{b \in A} e^{\hat{p}_b}}. \tag{5}$$

When used in MCTS, the probabilities are usually further normalized for legal actions, i.e.,

$$p_a = \frac{p'_a}{\sum_{b \in A(s)} p'_b}. \tag{6}$$

A real number $\hat{v}$ represents the value of a position, which is scaled to the range of [0, 1] by the sigmoid function [23],

$$v = \frac{1}{1 + e^{-\hat{v}}}. \tag{7}$$

From the collected self-play games, one position's policy $\hat{\boldsymbol{p}}$ and value $\hat{v}$ are updated as follows. Let $\boldsymbol{\pi}$ be the MCTS's policy based on visit counts and $z$ be the outcome of the self-play game, i.e., 1, 0.5, and 0 for a win, a draw, and a loss of the player to move. For policy, the goal is to have the estimated probability distribution $\boldsymbol{p}'$ (obtained from the lookup table) similar to $\boldsymbol{\pi}$. For value, the goal is to have the estimated value $v$ close to the expected outcome, i.e., the average of many $z$. Therefore, the loss functions for policy and value apply cross-entropy losses $(-\boldsymbol{\pi}^T \ln \boldsymbol{p}')$ and mean-squared error $(v - z)^2$, respectively, following Silver et al. [2]. By gradient decent, the policy and value in lookup tables are updated using

$$\hat{p}_a \leftarrow \hat{p}_a - \beta \cdot (p'_a - \pi_a) \tag{8}$$

and

$$\hat{v} \leftarrow \hat{v} - \beta \cdot (v - z) \cdot v \cdot (1 - v), \tag{9}$$

respectively, where $\beta$ is the learning rate. The derivations are shown by Hsueh [24] (cf. Chapter 4.2). Note that it is possible for the same position to have several different $\boldsymbol{\pi}$ and $z$, which are updated separately.

A synchronous version of the tabular AlphaZero, as presented in Algorithm 1, is used to train lookup tables. In the beginning, the policies and the values in the lookup table are initialized to random values. The lookup table is then incorporated into MCTS to collect $M$ self-play games. The most recent $K \times M$ games, if any, are used to optimize the lookup table. For example, with $K = 2$, the first iteration only uses $M$ games generated in that iteration to train, while the later iterations use $2M$. The process of self-play and optimization repeats for $N$ times in total.

**Algorithm 1** Tabular Synchronous AlphaZero

```
Initialize the lookup table
repeat
    Collect M self-play games
    Optimize by the most recent K × M games (if any)
until N iterations
```

## IV. EVALUATION METRICS OF LOOKUP TABLES

This section introduces three kinds of metrics to *directly* evaluate the lookup tables trained by the tabular AlphaZero. By direct, it means that the policies and values are employed to select actions or evaluate positions directly, without further incorporated into MCTS during evaluation as other

[2] $2 \times 4$ CDC action representation: 8 for flipping at a1, a2, ..., b4 and the remaining 32 for moving a1→a2, a1→a3, a1→a4, a1→b1, a2→a1, ..., b4→b2, b4→b3.

[3] EWN action representation: 6 combinations of two possibilities of the piece to move {≤ dice number, > dice number} and three possible directions {diagonal, horizontal, vertical}.

AlphaZero-related work often does. Note that MCTS in self-play and MCTS in evaluation are different, e.g., Dirichlet noise is not applied during evaluation. Finding promising hyper-parameters for MCTS in evaluation becomes another story. Assuming that the quality of MCTS's search results has a positive correlation with the quality of policies and values, the first two kinds of metrics measure the playing strength and the accuracy of the lookup tables. In addition, the exploration level of the state space is measured to investigate how hyper-parameters influence the diversity of self-play games.

The first kind of metric is the win rate playing against the optimal player, with players alternating taking the first turn [25]. The optimal player wins immediately when the position's theoretical value is 1 (100% win). Otherwise, the optimal player randomly selects one of the actions with the highest theoretical value. As for lookup tables, the policies are used to select the action with the highest probability among legal actions.[4] Namely, given a position $s$, the action to play is $\text{argmax}_{a\in A(s)} p_a$, where $A(s)$ is the set of legal actions at $s$ and $p_a$ the probability of selecting $a$. The win rate is denoted by $\text{WR}_\text{p}$, and those close to 50% are considered to approximate the optimal play. This metric aims to evaluate how well the learned policies can tell the best actions from others when playing against the optimal player. This paper excludes investigating how weak players can be defeated.

The second kind of metric aims to evaluate how accurate the policies and values are compared to the optimal play and theoretical values. This kind of metric is based on a set of test positions $S_{test} \subset S$, where $S$ is the state space. The positions selected for $S_{test}$ highly influence the results, and Subsection V-A will present the details about which positions are selected in this paper. For policies, the policy error, $\text{ERR}_\text{p}$, is defined as

$$\text{ERR}_\text{p} = \frac{\sum_{s\in S_{test}} \sum_{a\in A(s)} p_a \cdot (v^*_{a^*} - v^*_a)}{|S_{test}|}, \tag{10}$$

where $A(s)$ is the set of legal actions at position $s$, $p_a$ the normalized probability of legal action $a$, $a^*$ the best action, and $v^*_a$ the theoretical value of action $a$. In other words, $\text{ERR}_\text{p}$ calculates the drop of expected win rates assuming actions are selected according to the probability distributions from policies. Take as an example a position with three legal actions, where the theoretical values are 0.9, 0.7, and 0.2, respectively. Perfectly, a policy of (1, 0, 0) has an $\text{ERR}_\text{p}$ of 0 (i.e., 100% selecting the best action). The error becomes greater as the probabilities of the second or the third action increase, especially the third. For values, the mean absolute error, $\text{MAE}_\text{v}$, to the theoretical values is calculated, i.e.,

$$\text{MAE}_\text{v} = \frac{\sum_{s\in S_{test}} |v^*_s - v_s|}{|S_{test}|}, \tag{11}$$

where $v^*_s$ is the theoretical value of position $s$ and $v_s$ the lookup table's value.[5] $\text{ERR}_\text{p}$ and $\text{MAE}_\text{v}$ close to 0 indicate that the policies and values approximate the optimal and theoretical ones, respectively.

$\text{WR}_\text{p}$ and $\text{ERR}_\text{p}$ both evaluate the learned policies, but their results may not perfectly match (as will be seen in later experiments). Fundamentally, the two metrics employ policies in different ways. The player for collecting $\text{WR}_\text{p}$ selects the legal action that has the highest probability, focusing on measuring the ability to tell good actions from others. For $\text{ERR}_\text{p}$, the probabilities are involved in the calculation, aiming to measure how well the policy converges to the optimal play. $\text{WR}_\text{p}$ naturally evaluates the playing strength across opening games to endgames, while $\text{ERR}_\text{p}$ depends on test positions $S_{test}$. Both metrics provide useful information when evaluating the learning results.

The third kind of metric counts the number of positions that are updated at least $k$ times, where $k$ is a positive integer. This metric aims to evaluate the exploration level of the state space, also reflecting the diversity of positions in the training data. Assume that two different hyper-parameter settings are trained with the same number of self-play games. The setting has a higher exploration level of the state space if it has more updated positions. Different from the previous metrics that evaluate how good the learned policies/values are, this metric tries to verify how hyper-parameter settings influence the exploration during learning.

## V. EXPERIMENTS ON PUCT SELECTION HYPER-PARAMETERS

The following three sections will present three groups of experiments on hyper-parameters used in self-play games of the tabular AlphaZero. This section experiments on the hyper-parameters related to the PUCT algorithm, i.e., MCTS selection phase, including $c_{puct}$ in (1) and Dirichlet noise's $\alpha$ and $\varepsilon$ in (4). The experiment settings are described in Subsection V-A. The results of different $c_{puct}$, $\alpha$, and $\varepsilon$ settings are shown in Subsections V-B to V-D. Finally, an overall discussion is made in Subsection V-E.

### A. EXPERIMENT SETTINGS

Table 1 lists the selected game variants, including $\text{EWN}_{333}$, $\text{EWN}_{334}$, and $\text{EWN}_{343}$ for EWN, and $\text{CDC}_{\text{PPPP}}$, $\text{CDC}_{\text{KPPP}}$, and $\text{CDC}_{\text{GGCC}}$ for $2 \times 4$ CDC. The table also contains the average number of legal actions at each position, the average number of chance events at each position, the average game length, and the number of non-terminal positions. The

[4]Some readers may consider it better to also evaluate the value part by creating a player who selects the action with the highest expected value. Namely, the values serve as heuristics to look ahead one action, just as policies. However, a problem specific to lookup tables is that if the training on a position is insufficient, the value of the position remains close to the initialized 0.5, greatly affecting the action selection, especially when the best action has a theoretical value lower than 0.5.

[5]Even when the learned values deviate from the theoretical values, the player may still be able to play optimally when the expected value of the best action is higher than other actions. $\text{MAE}_\text{v}$ does not reflect this fact but is selected to investigate whether the tabular AlphaZero can approximate theoretical values.

**TABLE 1. Information about experimented game variants, including the average numbers of legal actions and chance events per position, the average game length, and the number of non-terminal positions, where W, H, #P for EWN mean board width, board height, and the number of initial pieces, respectively.**

| | Abbr. | Details | | | Avg. #actions /position | Avg. #chance events/position | Avg. game length | #Non-terminal positions |
|---|---|---|---|---|---|---|---|---|
| | | W | H | #P | | | | |
| EWN | $EWN_{333}$ | 3 | 3 | 3 | 2.9 | 3 | 5.3 | 367,956 |
| | $EWN_{334}$ | 3 | 3 | 4 | 2.4 | 4 | 5.3 | 6,316,032 |
| | $EWN_{343}$ | 3 | 4 | 3 | 2.7 | 3 | 7.8 | 3,268,620 |
| | | Piece set | | | | | | |
| 2×4 CDC | $CDC_{PPPP}$ | PPPP vs. pppp | | | 4.8 | 1.3 | 15.5 | 194,933 |
| | $CDC_{KPPP}$ | KPPP vs. kppp | | | 4.6 | 1.6 | 16.8 | 1,934,199 |
| | $CDC_{GGCC}$ | GGCC vs. ggcc | | | 4.4 | 1.7 | 18.3 | 4,032,500 |

former three were calculated from 1,000 self-play games by an MCTS with an $N_{sim}$ of 10,000 and random playouts, which was a reasonably strong player. The number of non-terminal positions means the total number of possible positions excluding terminal positions, i.e., positions may be learned.[6] Although EWN variants generally contain more non-terminal positions, 2 × 4 CDC variants have longer games and are expected to be more complex. The followings give some intuitions from the games' intrinsic properties. In EWN, pieces can only move toward the opponents' corners and never go back. For example, an $EWN_{333}$ game must end in fifteen actions (at the earliest three actions). In contrast, 2 × 4 CDC variants usually have longer games and require tactical strategies to capture opponents' pieces, which is expected to be more difficult.

For $EWN_{333}$, $EWN_{334}$, $EWN_{343}$, $CDC_{PPPP}$, $CDC_{KPPP}$, and $CDC_{GGCC}$, the learning lasted for 100, 1,600, 800, 300, 2,000, and 4,000 iterations, respectively (i.e., $N$ in Algorithm 1).[7] In each iteration, 1,000 self-play games were generated, and the optimization used 2,000 most recent games (if any), i.e., $M$ was 1,000 and $K$ was 2. The policies $\hat{p}_a$ and values $\hat{v}$ in lookup tables were initialized by a normal distribution with a mean of 0 and a standard deviation of 0.01. In this way, for each position, all actions had similar probabilities ($\approx 1/|A|$), and the value was close to 0.5 (win rate ≈50%). The learning rate $\beta$ was set to 1 at the beginning and then changed to 0.1 at the half (e.g., the 50th iteration for $EWN_{333}$) to stabilize the learning. The symmetry of positions was not exploited in the following experiments as Silver et al. [2] did, while the previous work on $CDC_{PPPP}$ made use of symmetry to augment training data [8], [24]. MCTS was single-threaded without parallelization, and the default values of hyper-parameters $c_{puct}$, Dirichlet $\alpha$ and $\varepsilon$, and $N_{sim}$ were 1, 1.5, 0.25, and 800, respectively. The default setting was run for several trials, and no substantial difference was obtained between different trials, as shown in Appendix A-A. Thus, the later experiments only run one trial for each setting.

At the end of each iteration, the lookup tables were evaluated by the metrics described in Section IV. (i) For $WR_p$, the lookup table player played 10,000 games against the optimal player, 5,000 games as the first player and the other 5,000 as the second. (ii) For $ERR_p$ and $MAE_v$, the test positions $S_{test}$ contained the initial positions for EWN ($EWN_{333}$: 108, $EWN_{334}$: 2,304, $EWN_{343}$: 108) and the positions within the first action for 2 × 4 CDC ($CDC_{PPPP}$: 17, $CDC_{KPPP}$: 33, $CDC_{GGCC}$: 33).[8] Although many of the positions are intrinsically identical, the learning on these positions is independent in this work (as Silver et al. [2] did not exploit the symmetry of positions either). These positions were selected since they appeared the most frequently in self-play games across various hyper-parameter settings. Errors might come from two totally different sources when employing lookup tables: too few updates for learning and wrong policies/values caused by far-from-optimal self-play games. The former problem may be simply solved by longer training, but the latter cannot. Thus, the latter is considered a more serious problem. To only focus on the latter, the opening positions (i.e., $S_{test}$ mentioned earlier) were the most feasible choice. (iii) For evaluating the level of exploration within the state space, the number of positions that were updated at least 1, 4, and 16 times were counted.

Two implementation details of MCTS that may influence the learning results are discussed as follows, though the issues are not limited to the MCTS in the AlphaZero algorithm. The first is tree nodes' initial values, where $W(s, a)/N(s, a)$ becomes undefined 0/0. Several kinds of commonly used implementations will be described and compared in Section VII. If not specified, the initial values were set to 0 as Silver et al. [2] did, as if the nodes had 100% losses [26]. The other implementation detail to discuss is chance nodes' expansion. Fig. 6 illustrates examples of MCTS expansion. In this work, when expanding a chance

[6] $EWN_{334}$ has a similar number of legal actions and game length to $EWN_{333}$ but has about 17 times non-terminal positions. The reason is that $EWN_{334}$ has one more initial piece for both players, resulting in more variants of gameplay than $EWN_{333}$.

[7] The numbers were selected considering each game variant's complexity with some preliminary experiments. Such many iterations were sufficient for good settings to learn, as can be seen in training curves such as Fig. 7.

[8] In more detail, 108 for $EWN_{333}$ and $EWN_{343}$ comes from 3! × 3! × 3, where 3! is for red pieces' permutation, another 3! for blue pieces, and the 3 for dice numbers 1 to 3; 17 for $CDC_{PPPP}$ comes from 1+8 × 2, containing 1 initial position, where all pieces are unrevealed, and 16 positions after the first flipping action (8 candidate squares and 2 possible outcomes, red and black pawns); similarly for the remaining.

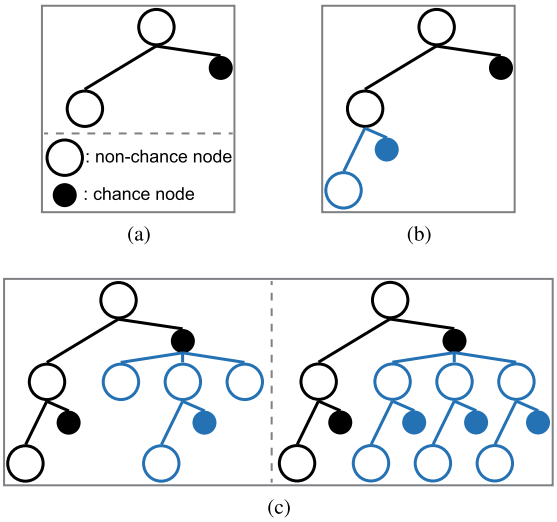



**FIGURE 6. Examples of MCTS expansion with chance nodes, (a) a non-chance node with two leaf children where one is a non-chance node and the other a chance node, (b) expanding the non-chance node with its legal actions, and (c) expanding the chance node with its chance events, and further expanding a chance event (left) or all chance events (right).**

node, a chance event (e.g., an unrevealed piece for CDC and a dice number for EWN) was randomly sampled according to the probability distribution. The child node corresponding to the sampled chance event was further expanded with legal actions (Fig. 6c (left)). In this way, each simulation expands a non-chance node unless the game ends. Another possible implementation was to expand all chance events in the same simulation (Fig. 6c (right)), requiring more computation. The comparison of different implementations is out of this paper's scope.

### B. RESULTS OF DIFFERENT $c_{puct}$ VALUES

The first experiment was on $c_{puct}$, where Silver et al. [2] did not specify the value they used, and Tian et al. [9] used 1.5 in their experiments on Go. A wide range of thirteen $c_{puct}$ settings was tested, including 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8, 16, 32, and 64. Fig. 7 depicts the $WR_p$ training curves for the six experimented game variants. Only about half of the $c_{puct}$ settings are shown for better display. In the six game variants, except for two extreme ends of $c_{puct}$, $WR_p$ got close to 50%, meaning that the learned policies could generally distinguish good actions. In addition, it was observed that higher $c_{puct}$, meaning higher levels of exploration, tended to learn slower.

Fig. 8 shows the $WR_p$ of the last iteration with 95% confidence intervals (all within $\pm 1\%$). The maximum $WR_p$ during training is also included.[9] Generally, the last and the max did not differ too much, and $c_{puct}$ values around 1 obtained $WR_p$ close to 50% in the six game variants, though the proper ranges of $c_{puct}$ seemed to become narrower for more complex game variants. The reasons for having poor performances with too-high and too-low $c_{puct}$ were different. For too-low ones, it was related to the implementation of tree node initialization and will be discussed in more detail in Section VII. For too-high ones, the PUCT algorithm put too much emphasis on exploring less-visited actions, making each action have a close number of visits. Note that for the policies to learn and for action selection in self-play games, the probability distributions were in proportion to the visit counts $N(s_0, a)$. Thus, too-high $c_{puct}$ made the policies and self-play games close to random play and failed to learn well.

Fig. 9 shows the last iterations' $ERR_p$ and $MAE_v$ calculated on the test positions $S_{test}$ for $EWN_{333}$ and $CDC_{PPPP}$. The y-axis has different ranges for each game variant, which are best to show the tendencies. The results of the remaining four game variants have similar tendencies and are put in Appendix A-B. Such a presentation will be applied to the rest of this paper unless there are discussions on the six game variants. For low and high values of $c_{puct}$, both $ERR_p$ and $MAE_v$ went high. On the one hand, when $c_{puct}$ was too low, the MCTS in self-play games often wrongly concentrated the simulations on non-optimal actions, which will be explained in Section VII. Thus, the learned policies gave high probabilities to non-optimal actions, making the $ERR_p$ high. Also, the outcomes of self-play games were wrongly biased by the non-optimal actions, resulting in high $MAE_v$. On the other hand, when $c_{puct}$ was too high, MCTS distributed the simulations almost evenly to the actions. The resulting policies and self-play games were close to random play, making the $ERR_p$ and $MAE_v$ high.

Fig. 10 plots the numbers of positions updated $\geq 1$, $\geq 4$, and $\geq 16$ times during the whole training for $EWN_{333}$ and $CDC_{PPPP}$ with different $c_{puct}$. The curves of $\geq 1$, $\geq 4$, and $\geq 16$ showed similar tendencies. Generally, with a higher $c_{puct}$, the number of updated positions went higher, which was as expected since higher $c_{puct}$ led to higher levels of exploration during search and thus more diverse positions in self-play games. Appendix A-B will make more detailed discussions on exceptional behaviors (low $c_{puct}$ for $EWN_{333}$ and high $c_{puct}$ for $CDC_{PPPP}$) that were game-specific.

### C. RESULTS OF DIFFERENT DIRICHLET $\alpha$ VALUES

For Dirichlet noise, the hyper-parameter $\alpha$ determines the concentration level of the distribution, and lower $\alpha$ leads to more exploration, as discussed in Subsection II-A. Silver et al. [2] specially adjusted $\alpha$ for different games according to the average number of legal actions. The values they used for Go, shogi, and chess were 0.03, 0.15, and 0.3, respectively. Games with fewer legal actions received higher $\alpha$. Since all the experimented game variants in this paper have much fewer legal actions, six higher $\alpha$, 0.75, 1.5, 3, 6, 9, and 12, were also tried. In the six game variants, different $\alpha$ settings did not differ too much. All had the last $WR_p$ close to 50% and relatively low $ERR_p$ and $MAE_v$. Thus, the detailed results are omitted. The reason was suspected to be the low

[9]When applying the AlphaZero algorithm to more complex games with DNNs, it is computationally expensive to sample DNNs periodically during training and evaluate the DNNs thoroughly. Thus, in this paper's analyses, the performance of the last iterations was considered more important.

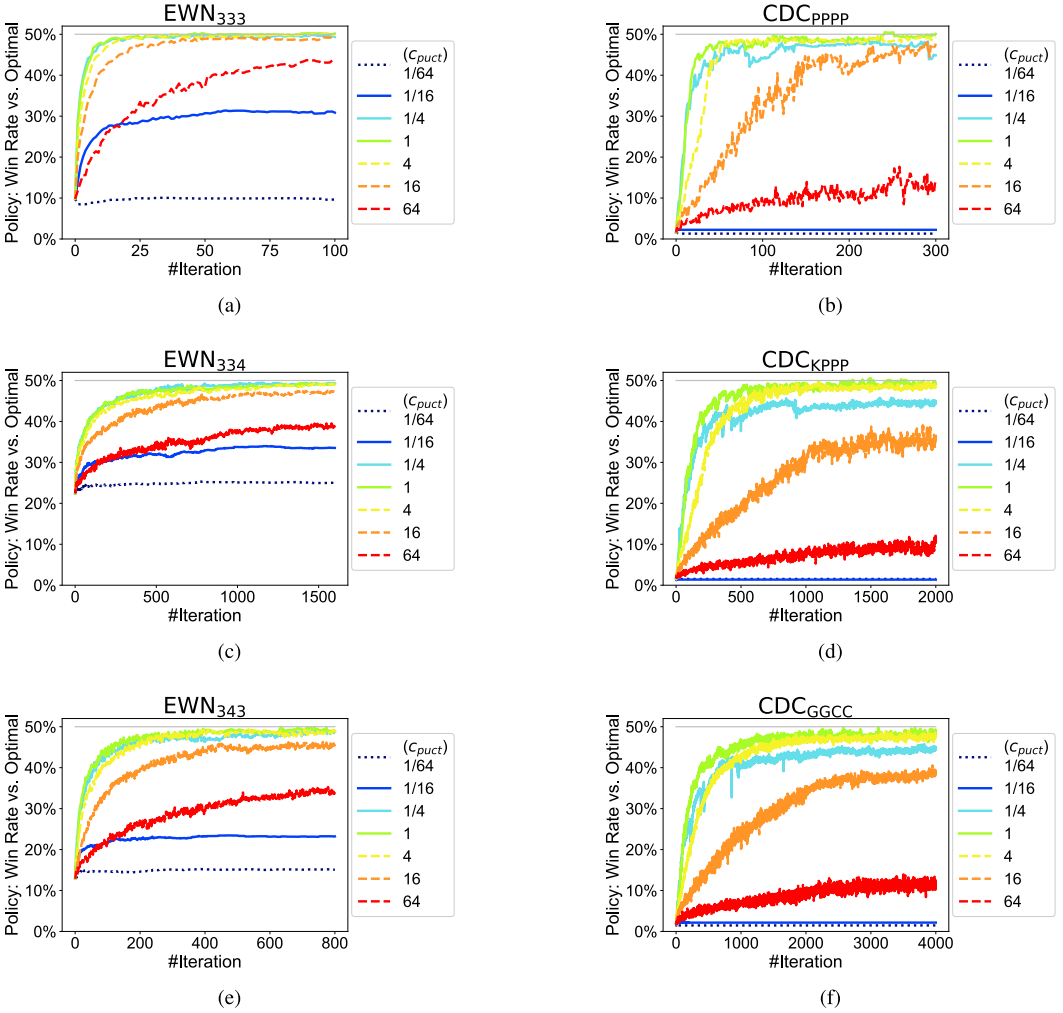



**FIGURE 7.** $WR_p$ training curves with different $c_{puct}$ for the six selected game variants (left: EWN, right: 2 × 4 CDC).

numbers of legal actions in the experimented game variants compared to Go, shogi, and chess.

### D. RESULTS OF DIFFERENT DIRICHLET $\varepsilon$ VALUES

For Dirichlet noise, the hyper-parameter $\varepsilon$ determines the weight of the noise (4). A total of nine values were tested for $\varepsilon$, including 0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, and 0.5. Among these values, 0 meant ignoring the Dirichlet noise, and 0.25 was used by Silver et al. [1], [2]. Fig. 11 plots the last and the maximum $WR_p$ for the six experimented game variants. With an $\varepsilon$ greater than 0, the $WR_p$ was close to 50% in the six game variants, showing that the weight of the noise did not influence too much. An $\varepsilon$ of 0 learned well in $EWN_{333}$ and $EWN_{334}$ but got relatively bad results in the remaining four game variants. The results of $EWN_{333}$ and $EWN_{334}$ were considered exceptions, explained as follows. These two game variants had the shortest average game length, and the average numbers of legal actions and chance events at each position were relatively low. It is possible for MCTS to reach terminal positions and find good actions without the assistance of Dirichlet noise, which is not the case for most of the other games that are more complex.

Fig. 12 shows $EWN_{333}$ and $CDC_{PPPP}$'s $ERR_p$ and $MAE_v$ of the last iterations. Except for the settings that failed to learn the optimal play (e.g., an $\varepsilon$ of 0 for $CDC_{PPPP}$), the $ERR_p$ and $MAE_v$ were relatively low, meaning that the policies and values were accurately learned to some extent. Fig. 13 depicts the numbers of positions with $\geq$1, $\geq$4, and $\geq$16 updates for $EWN_{333}$ and $CDC_{PPPP}$. As expected, the numbers increased as $\varepsilon$ increased.

### E. SUMMARY

The experimented hyper-parameters in this section, $c_{puct}$ and Dirichlet noise's $\alpha$ and $\varepsilon$, all related to the exploration term in the PUCT algorithm, i.e., the second term in (1). Dirichlet noise was added only to the root's actions, while $c_{puct}$ was used by all nodes. Higher $c_{puct}$, lower $\alpha$, and higher $\varepsilon$ led to higher levels of exploration. Generally, with higher levels of exploration, the numbers of updated positions increased (Figs. 10 and 13).

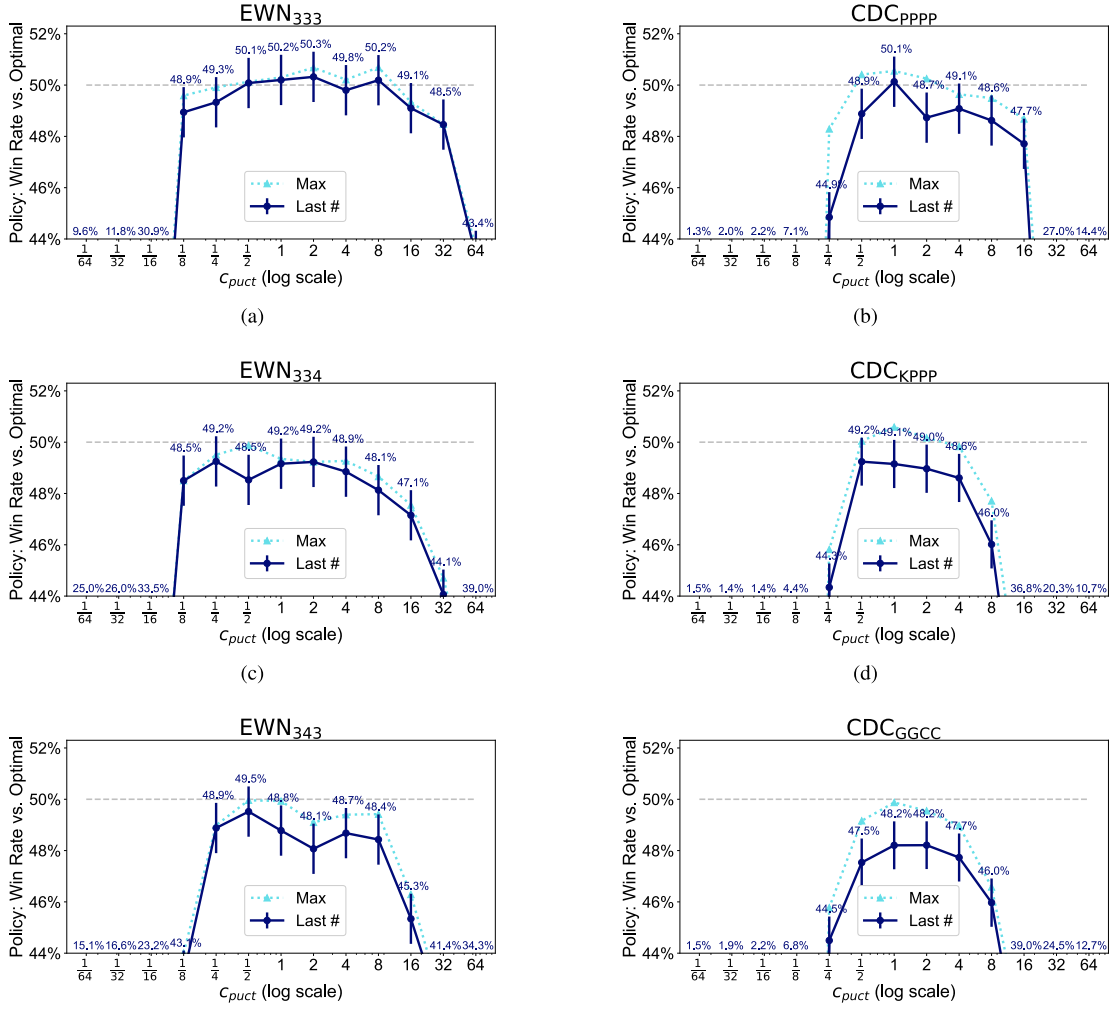



**FIGURE 8.** Last and max $WR_p$ with different $c_{puct}$ for the six selected game variants (left: EWN, right: 2 × 4 CDC).

In the experiments, the ranges of hyper-parameter settings were designed to be wide in order to find the ranges in which the tabular AlphaZero could learn near-optimal actions and accurate policy and value outputs. The influence of each individual hyper-parameter was investigated in this study. How the hyper-parameters might influence each other was not in the scope.

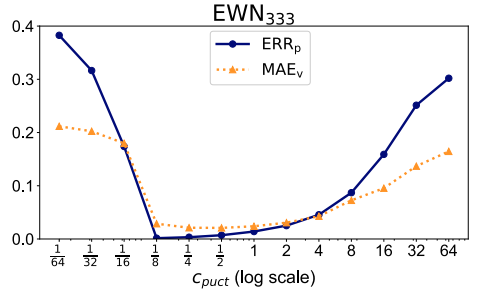
Many of the tested settings had $WR_p$ close to 50% (Figs. 8 and 11), demonstrating the robustness of the tabular AlphaZero on learning near-optimal actions. Such settings included $c_{puct}$ between 1/2 to 4, Dirichlet $\alpha$ between 0.03 to 12, and Dirichlet $\varepsilon$ between 0.1 and 0.5 for all the six game variants. For these settings, the $ERR_p$ and $MAE_v$ roughly increased as the levels of exploration increased (Figs. 9 and 12). As the discussions on Fig. 9 in Subsection V-B (the 3rd paragraph), higher levels of exploration made MCTS select actions more evenly. Thus, the action selections in self-play games and the policies in training data got closer to random play, resulting in higher $ERR_p$ and $MAE_v$.

## VI. EXPERIMENTS ON SIMULATION NUMBERS

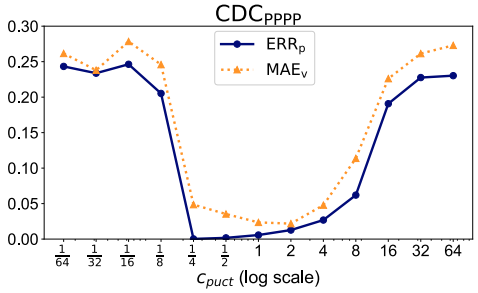
This section investigates how the learning of tabular AlphaZero is influenced by the number of simulations ($N_{sim}$) in the MCTS for self-play games. On the one hand, many studies in the past have shown that more simulations made the MCTS players stronger [9], [17], [25], [27]. However, too-high numbers are not only extremely costly, especially in real-world games such as Go and chess, but also not guaranteed to generate proper self-play games for learning. On the other hand, too-low numbers are less computationally consuming but are expected to make the MCTS players weak, playing many bad actions during self-play. The role of the MCTS in self-play games of the AlphaZero algorithm is more to generate proper training data in a reasonable time than just play well or quickly. Subsections VI-A and VI-B present the experiment settings and results, respectively.

### A. EXPERIMENT SETTINGS

A wide range of $N_{sim}$ settings were tested, including 6, 12, 25, 50, 100, 200, 400, 800, 1,600, 3,200, and 6,400.

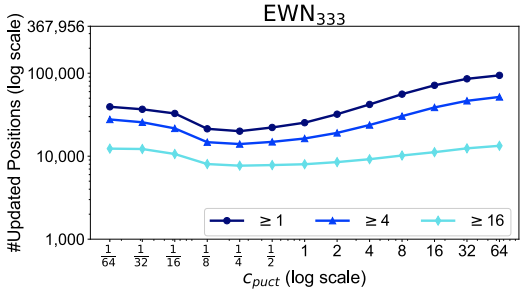



(a)

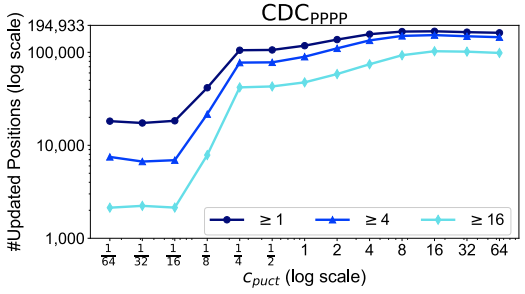



(b)

**FIGURE 9.** $\mathbf{ERR_p}$ **and** $\mathbf{MAE_v}$ **of the last iterations with different** $\boldsymbol{c_{puct}}$ **for (a)** $\mathbf{EWN_{333}}$ **and (b)** $\mathbf{CDC_{PPPP}}$**.**




(a)




(b)

**FIGURE 10.** **The number of positions updated ≥1, ≥4, and ≥16 times during training for (a)** $\mathbf{EWN_{333}}$ **and (b)** $\mathbf{CDC_{PPPP}}$ **with different** $\boldsymbol{c_{puct}}$**.**

For $EWN_{333}$, $EWN_{343}$, and $CDC_{PPPP}$ that required fewer computation resources, 12,800 and 25,600 were also tried. Among the values, Silver et al. [1] and Tian et al. [9] used 1,600 for Go. Silver et al. [2] further reduced it to 800 when applying to Go, shogi, and chess.

Two experiments were conducted to analyze $N_{sim}$ from different aspects. One had the same number of self-play games per iteration, and the other had the same computation time to do self-play. In more detail, the former played 1,000 games in each iteration, as in Section V. Although higher $N_{sim}$ took more time, this experiment aimed to exclude the factor of computation resources. The latter experiment tried to make the computation resources even and spent the same time doing self-play (i.e., $T_{self}$ seconds per iteration). Two game variants were selected for this experiment, $EWN_{333}$ and $CDC_{PPPP}$, whose $T_{self}$ were 2.143 and 18.418, respectively. The time settings were determined by the average of all iterations playing 1,000 games with an $N_{sim}$ of 800 on machines equipped with Intel (R) Core (TM) i9-10900K CPU @ 3.70 GHz.

For both experiments, games from the two most recent iterations ($K = 2$) were used in optimization. This experiment was expected to provide insights into practical training with limited computation resources. The remaining settings of the two experiments, including the evaluations, were the same as the default values in Subsection V-A.

### B. EXPERIMENT RESULTS

Fig. 14 shows the $WR_p$ training curves for $EWN_{333}$ and $CDC_{PPPP}$ with different $N_{sim}$ under 1,000 games per iteration. The results of the remaining four game variants are put in Appendix A-B. With too-low $N_{sim}$, the tabular AlphaZero failed to approximate the optimal play, which will be discussed more thoroughly in Section VII. With higher $N_{sim}$, the curves grew more drastically and *soon* converged to 50%. The results were reasonable in that higher $N_{sim}$ was expected to produce better policies and play self-play games closer to the optimal play.

Fig. 15 contains $EWN_{333}$ and $CDC_{PPPP}$'s $WR_p$ of the last iterations and the maximum, trained using different $N_{sim}$ under 1,000 games per iteration. The 95% confidence intervals were within ±1%. The results suggested that an adequate $N_{sim}$ (e.g., 25 for $EWN_{333}$ or 50 for $CDC_{PPPP}$) could already learn approximately optimal play. Compared to chess, shogi, or Go, the experimented game variants have smaller game trees, so it was unsurprising that an $N_{sim}$ of 50 worked well.

As for the results of fixing the self-play time to $T_{self}$ per iteration, first, Fig. 16 plots the average numbers of games played per iteration with different $N_{sim}$. When $N_{sim}$ was as low as 6 or 12, the number of played games was lower than expected compared to other $N_{sim}$ settings. A possible reason was that the overhead for creating new search trees or games was non-negligible. Another possible reason was that the MCTS was more likely unable to select good actions, making the games longer. For $N_{sim}$ higher than 12, the decreasing tendencies for the average number of played games were stable. When $N_{sim}$ was 25,600, only about 40 games were played in each iteration.

The $WR_p$ training curves ($T_{self}$ self-play time per iteration) are depicted in Fig. 17. Too-low $N_{sim}$ (e.g., 6) still failed to

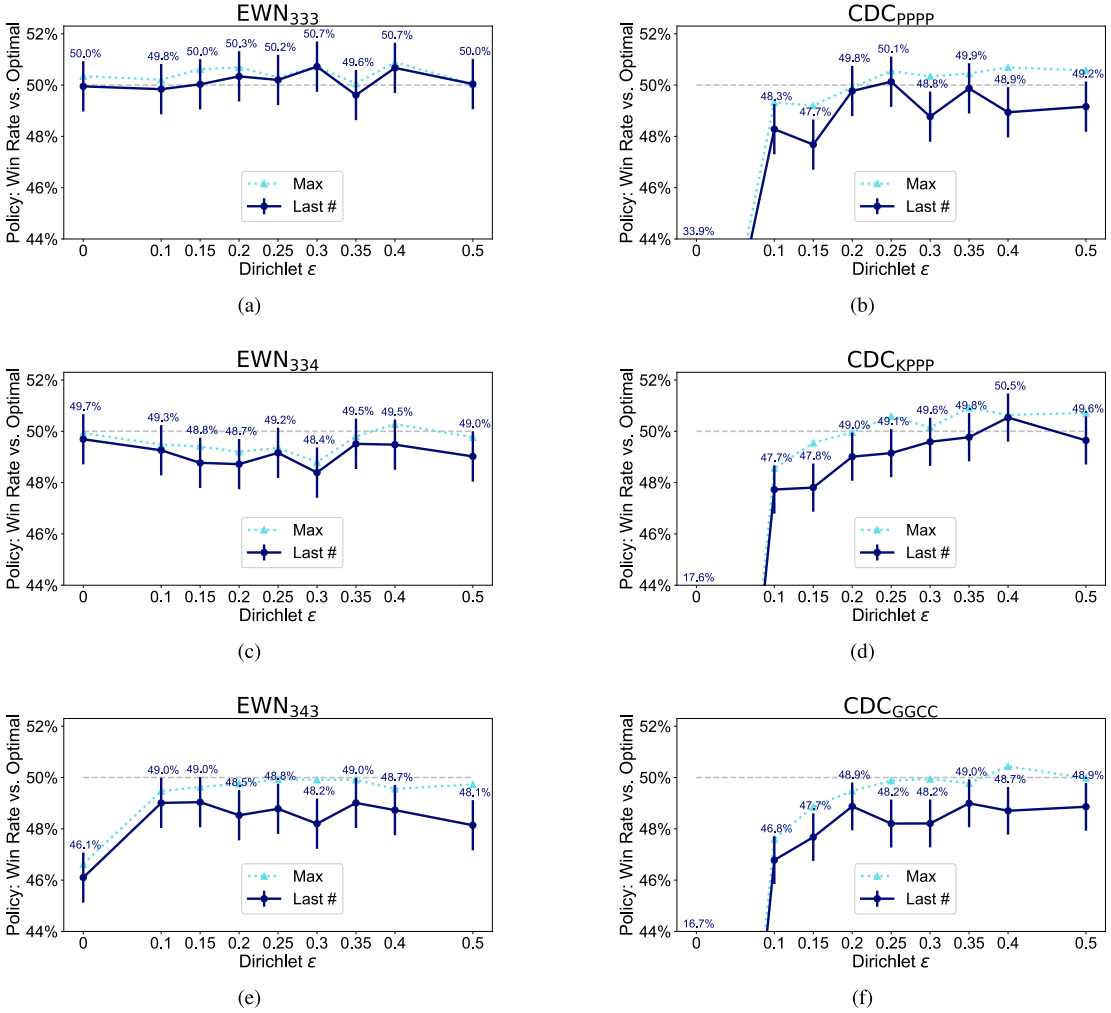



**FIGURE 11.** Last and max $WR_p$ with different $\varepsilon$ for the six selected game variants (left: EWN, right: 2 × 4 CDC).

learn, even with more played games. When $N_{sim}$ was high, the learning was slow as expected since only a few self-play games could be generated within the limited time, making the training data insufficient.[10] When considering both the learning speed (wall-clock time) and the final win rates, $N_{sim}$ between 50 and 400 seemed promising for $EWN_{333}$ and $CDC_{PPPP}$.

Fig. 18 shows the last iterations' $ERR_p$ and $MAE_v$ calculated on the test positions $S_{test}$ for $EWN_{333}$ and $CDC_{PPPP}$ with different $N_{sim}$ under 1,000 games per iteration. Those under the same self-play time are omitted since different $N_{sim}$ had different numbers of played games, introducing more influencing factors. Generally, $ERR_p$ and $MAE_v$ decreased as $N_{sim}$ increased, showing that more simulations helped the tabular AlphaZero learn policies and values more accurately, especially policies.

Fig. 19 plots the numbers of positions updated $\geq$1, $\geq$4, and $\geq$16 times during the whole training for $EWN_{333}$ and $CDC_{PPPP}$ with different $N_{sim}$ under the same number of self-play games. With sufficient $N_{sim}$ (say $\geq$ 100), the numbers steadily decreased as $N_{sim}$ increased, explained as follows. When MCTS had more simulations, the visit counts usually went to the most promising actions (from its view). Since actions in self-play games were selected in proportion to the visit counts, the probabilities of selecting less-visited actions were decreased, lowering the level of exploration.

To sum up, the experiments demonstrated that a moderate $N_{sim}$ was sufficient for tabular AlphaZero's learning (e.g., 50 to 400 when fixing self-play time). With a too-low $N_{sim}$, it was as expected that learning failed because good actions were hardly found. With a too-high $N_{sim}$, it took a very long time to generate self-play games, making the learning inefficient. In addition, it was observed that high $N_{sim}$ tended to have lower levels of exploration in

[10] In addition to fixing the self-play time, another way to reduce computation resources for high $N_{sim}$ was to remain 1,000 games per iteration but decrease the total iterations. For example, $CDC_{PPPP}$ had the final $WR_p$ of 50.1% when trained for 300 iterations with an $N_{sim}$ of 800. For an $N_{sim}$ of 25,600, 13 iterations cost about the same time, but the $WR_p$ was 41.7%. The results again showed that too-high $N_{sim}$ obtained worse $WR_p$ under a similar cost of computation resources.

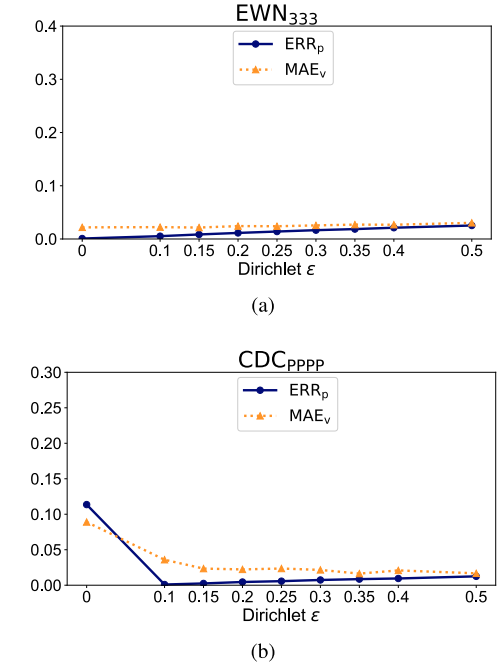



**FIGURE 12.** $ERR_p$ and $MAE_v$ of the last iterations with different $\varepsilon$ for (a) $EWN_{333}$ and (b) $CDC_{PPPP}$.

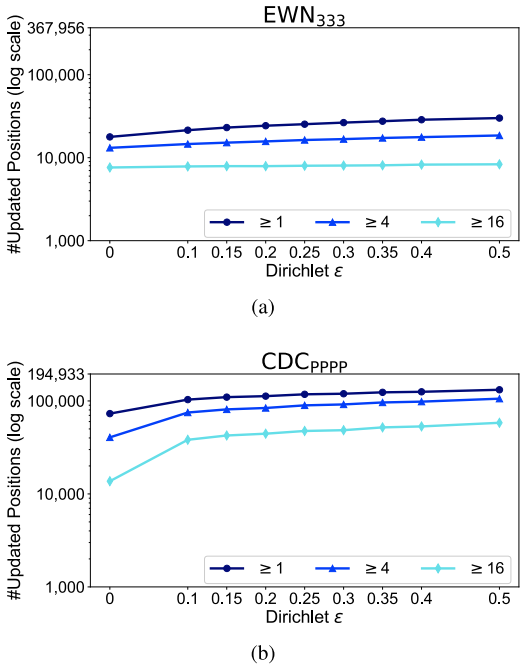



**FIGURE 13.** The number of positions updated ≥1, ≥4, and ≥16 times during training for (a) $EWN_{333}$ and (b) $CDC_{PPPP}$ with different $\varepsilon$.

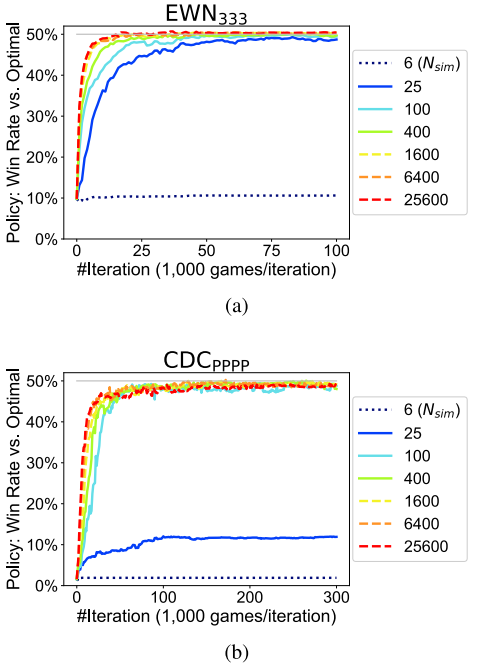



**FIGURE 14.** $WR_p$ training curves with different $N_{sim}$ under 1,000 games/iteration for (a) $EWN_{333}$ and (b) $CDC_{PPPP}$.

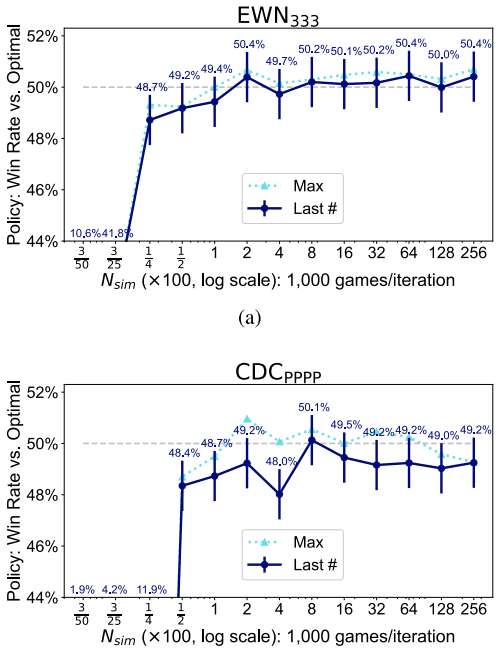



**FIGURE 15.** Last and max $WR_p$ with different $N_{sim}$ for (a) $EWN_{333}$ and (b) $CDC_{PPPP}$ under 1,000 games/iteration.

the state space. It remains an open question how such a phenomenon influences the learning of the AlphaZero algorithm on complex games with feature extractors such as DNNs.

## VII. EXPERIMENTS ON TREE NODE INITIALIZATION

Tree node initialization is an important topic for MCTS, not limited to the AlphaZero algorithm nor related to whether the game is deterministic or stochastic. The question is, what values should be assigned to $W(s, a)/N(s, a)$ when the action has not been tried? A possible implementation was to select untried actions first [28], equivalent to setting the initial $W/N$ to $+\infty$. However, when applying to games with many legal actions such as Go, the search is inefficient in trying many unimportant actions. Some other implementations tried

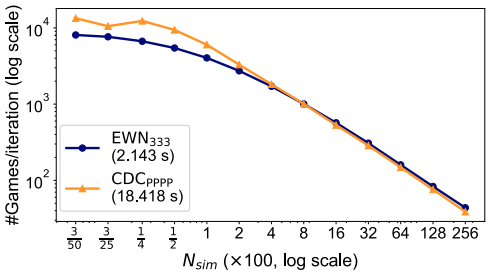
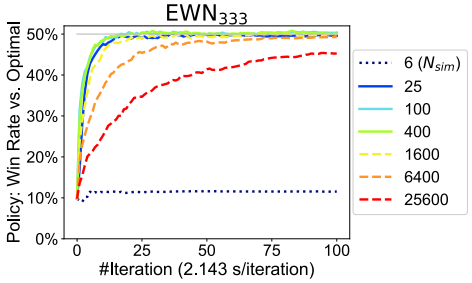
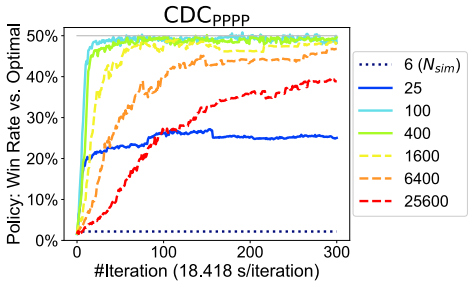



**FIGURE 16.** $\text{EWN}_{333}$ and $\text{CDC}_{\text{PPPP}}$'s average #games/iteration with different $N_{sim}$.




(a)



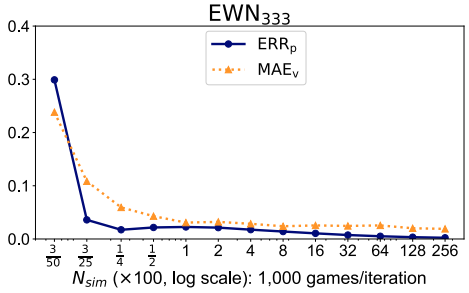

(b)

**FIGURE 17.** $\text{WR}_{\text{P}}$ training curves with different $N_{sim}$ for (a) $\text{EWN}_{333}$: 2.143 s/iteration and (b) $\text{CDC}_{\text{PPPP}}$: 18.418 s/iteration.

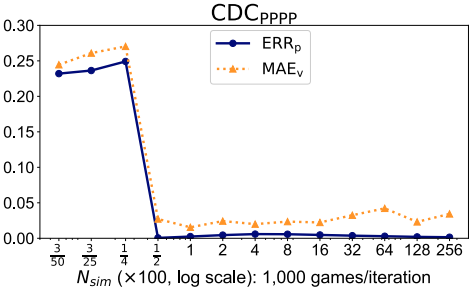
to focus more on a few promising actions at each node. Silver et al. [2] set the initial $W/N$ to $+0$, meaning a loss [26]. An author of the paper suggested that $+0$ seemed to work the best for the AlphaZero algorithm [29]. Another possible tree node initialization was to assume each state to be an even game and assign $W/N$ to $+0.5$ [30]. Using heuristic evaluation functions to initialize $W/N$ was also proposed [30] but might not be suitable for the AlphaZero algorithm, explained as follows. The algorithm does not assume the use of additional heuristics. Although the value network can serve as such heuristics, using it to initialize the $W/N$ of all untried actions is computationally expensive.

This section experiments on four possibilities, $+0$ (loss), $+0.5$ (even), $+1$ (win), and $+\infty$ (untried first), investigating how the learning of tabular AlphaZero is influenced.[11] The comparisons were made using different $N_{sim}$ and $c_{puct}$. The experiment settings were the same as those described

[11]The results may be different for the MCTS in evaluation and are out of this paper's discussions, as mentioned in the 1st paragraph of Section IV.

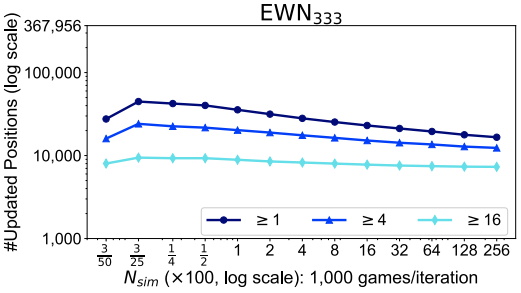



(a)

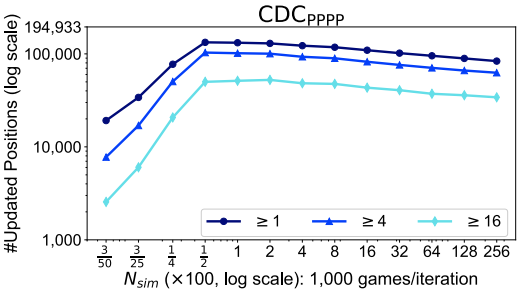



(b)

**FIGURE 18.** $\text{ERR}_{\text{P}}$ and $\text{MAE}_{\text{V}}$ of the last iterations with different $N_{sim}$ under 1,000 games/iteration for (a) $\text{EWN}_{333}$ and (b) $\text{CDC}_{\text{PPPP}}$.



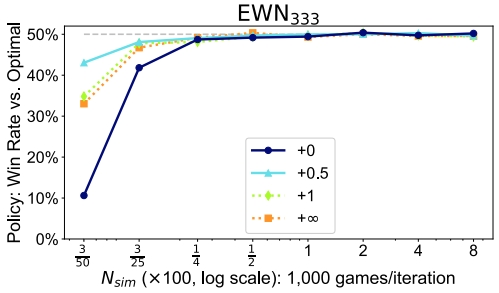

(a)



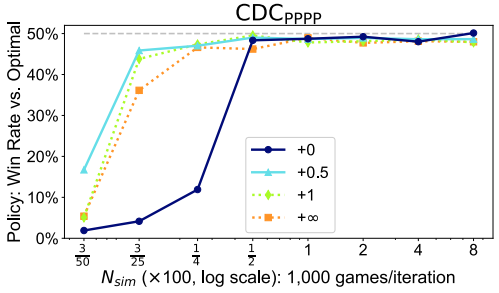

(b)

**FIGURE 19.** The number of positions updated ≥1, ≥4, and ≥16 times during training for (a) $\text{EWN}_{333}$ and (b) $\text{CDC}_{\text{PPPP}}$ with different $N_{sim}$ under 1,000 self-play games/iteration.

in Subsection V-A except for the initial $W/N$ and the corresponding $N_{sim}$ or $c_{puct}$. It was observed that the results of low $N_{sim}$ and low $c_{puct}$ differed considerably between different ways to initialize tree nodes, discussed in Subsections VII-A and VII-B, respectively.




(a)




(b)

**FIGURE 20.** Last $WR_p$ of different initial $W/N$ for (a) $EWN_{333}$ and (b) $CDC_{PPPP}$ with different $N_{sim}$.

### A. COMPARISON BASED ON $N_{sim}$

In the experiments, eight $N_{sim}$ settings were tested, including 6, 12, 25, 50, 100, 200, 400, and 800. Fig. 20 shows the $WR_p$ of the last iterations for $EWN_{333}$ and $CDC_{PPPP}$. With sufficiently high $N_{sim}$ (say 25 for $EWN_{333}$ and 100 for $CDC_{PPPP}$), the last $WR_p$ did not differ too much, and all were close to 50%. When further looking into the $WR_p$ training curves, i.e., the right-most plots in Fig. 21, different settings of initial $W/N$ did not differ too much, either.

With an $N_{sim}$ of 6, the last $WR_p$ failed to reach 50%. In more detail, an initial $W/N$ of $+0.5$ was the closest (43.05% for $EWN_{333}$ and 16.72% for $CDC_{PPPP}$), $+1$ and $+\infty$ slightly worse (about 34% for $EWN_{333}$ and about 5% for $CDC_{PPPP}$), and $+0$ clearly the worst (10.62% for $EWN_{333}$ and 1.90% for $CDC_{PPPP}$), also shown in the training curves in Fig. 21 (left-most). Interestingly, different implementations had different reasons for the inability to learn well. Although $N_{sim}$ of 6, 12, etc. may seem obviously bad settings, especially when considering complex games such as Go, it is still worth investigating the reasons behind the failure, explained as follows. It is suspected that the minimum required $N_{sim}$ relates to the number of legal actions and the portion of good actions among legal ones. Thus, when applying to other games, higher $N_{sim}$ (e.g., 100) may still fail to learn well for similar reasons discussed in this subsection.

For $+\infty$ and $+1$, the reasons were related to trying all actions exhaustively. In the case of $+\infty$, MCTS always selected untried actions first. When $N_{sim}$ was just sufficient to try legal actions at the root once, each of the actions had the same probability of being selected.[12] With a slightly higher $N_{sim}$ (e.g., 6 for $EWN_{333}$ and 12 for $CDC_{PPPP}$), the MCTS could gradually spend more simulations on better actions. Note that the probabilities of selecting actions in self-play games were in proportion to the visit counts. Even if an action was selected only once, with a low $N_{sim}$, the probability was still high. Although the action selection mechanism was originally introduced to increase the diversity of self-play games, in this case, the action selection was closer to random play similar to too-high $c_{puct}$, which also failed to learn well, as shown in Fig. 7. When the initial $W/N$ was $+1$, the effect was similar to $+\infty$, i.e., likely to select untried actions first. A slight difference occurred when winning actions were found. Since a winning action's $W/N$ after update became $+1$, equal to those of untried actions, the action would be selected again if its exploration term was still the highest, making the implementation of $+1$ slightly better than $+\infty$.

For $+0.5$ and $+0$, the reasons were related to the initial policies and values, e.g., the normal distribution with a mean of 0 and a standard deviation of 0.01 described in Subsection V-A. Since neural networks are usually initialized in similar ways, the following discussions are suspected to be applicable to the general AlphaZero algorithm, which will be confirmed by the results in Section VIII. Consider the self-play games in the first iteration that used the randomly initialized lookup tables (or neural networks). For a tree node $s$ that none of the actions have been visited yet, the actions' PUCT scores (1) only differed in the prior probability $P(s, a)$. After an action $a$ was selected and updated, the action's $W/N$ became $v \approx 0.5$ from the lookup table.[13]

In the case of $+0.5$, almost all actions were selected evenly during searches in the first few iterations since different actions had close PUCT scores, no matter tried or untried. More specifically, the $+0.5$ did not differ too much from the randomly initialized $v \approx 0.5$, and all actions had similar initial probabilities. For example, the $EWN_{333}$ position in Fig. 4a with a dice number of 2 has three legal actions ↘ (b1→c2), → (b1→b2), and ↓ (b1→c1), whose theoretical values are 0.827, 0.193, and 0.144, respectively. The three actions' initial probabilities were all close to 1/3, as can be seen in the plots in Fig. 22. The left-most plot of Fig. 22 shows the learned policies for this position with an $N_{sim}$ of 6 during training. In the first few iterations, the three actions had similar probabilities, and then the best action ↘ soon surpassed the other two and converged to almost 100%. This was because the tabular AlphaZero already found action ↘ to be advantageous (better than 0.5) and could concentrate on this action. The behavior was suspected of making $+0.5$ the best setting when $N_{sim}$ was extremely low.

[12] Although an $N_{sim}$ of 6 was higher than the average number of legal actions in Table 1, it was still insufficient to try all actions in some positions. For example, the initial position in $2 \times 4$ CDC has 8 legal actions, requiring 8+1 simulations to try all actions once (1 for expanding the root).

[13] Silver et al. [2] shifted the networks' estimated values in $[-1, 1]$ to $[0, 1]$ in the search [26]. Thus, $0 \in [-1, 1]$ is equivalent to $0.5 \in [0, 1]$.

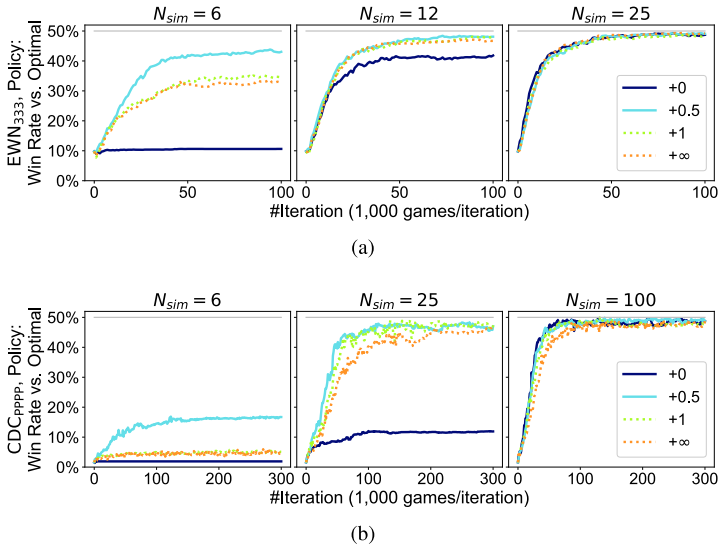



**FIGURE 21.** $WR_p$ training curves of different initial $W/N$ for (a) $EWN_{333}$ and (b) $CDC_{PPPP}$ with selected $N_{sim}$.

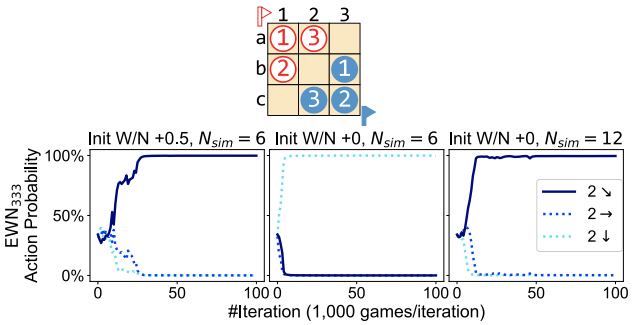



**FIGURE 22.** The position Fig. 4a assuming red's turn with a dice number of 2 and the learned policy with different $N_{sim}$ and initial $W/N$.

Finally, in the case of +0, for the same $EWN_{333}$ example with an $N_{sim}$ of 6, almost all probabilities went to action ↓, which was the theoretically worst action. The following explains why this could have happened. With a randomly initialized lookup table, once an action (in this case ↓) was selected during search, its $W/N$ was updated to $v \approx 0.5$, while others remained +0. To select other actions at the root, the exploration term (the second term in (1)) alone needed to exceed the PUCT score of the already-selected action ($v \approx 0.5$ + its exploration term), which was difficult in a few simulations unless very high $c_{puct}$. When $N_{sim}$ was 6, the learned policies might be wrongly biased by the randomly initialized policies and values (and Dirichlet noise), as shown in the center plot of Fig. 22. The self-play games were also wrongly biased, and thus the learning failed. The problem was alleviated soon as $N_{sim}$ increased. The right-most plot of Fig. 22 depicts the results of the same $EWN_{333}$ example with an $N_{sim}$ of 12. In the first few iterations, the three actions had similar probabilities, learned from the similar simulation numbers of the MCTS in self-play games. As the learning progressed, the probability of selecting the best action soon converged close to 100%.

When applying to more complex games, one more thing worth discussing is that Silver et al. [1], [2] employed multi-threaded MCTS and a technique called virtual loss [31] to ensure different threads traverse the search tree differently. With virtual losses, each action upon traversal increased its $N(s, a)$ by a given number, subtracted back in backpropagation. It is suspected that virtual losses also helped overcome the potential problem caused by randomly initialized policies and values under the implementation of +0. In contrast, for +0.5, virtual losses may make untried actions more likely to be selected. This may be one of the reasons why +0 was reported to be the best [29].

### B. COMPARISON BASED ON $c_{puct}$

In the experiments, the thirteen $c_{puct}$ settings as Subsection V-B were tested. Fig. 23 depicts the $WR_p$ of the last iterations for $EWN_{333}$ and $CDC_{PPPP}$. Considerable differences were observed when $c_{puct}$ was extremely low (say $\leq 1/16$), where +0 was the worst and the remaining three were all close to 50%. The reason that +0 failed to learn with too low $c_{puct}$ was also related to the initial policies and values. Consider the first iteration searching with randomly initialized lookup tables. When an action $a$ at a node $s$ was selected and the rest not, $a$ was the only action having $W/N \approx 0.5$ while the rest +0. When $c_{puct}$ was too low, MCTS kept selecting $a$ even with an $N_{sim}$ of 800. Similar to Fig. 22 (center plot), the self-play games were wrongly biased by randomness, making the learning fail.

When $W/N$ was +0.5, +1, or $+\infty$, each action at the root was tried at least several times so that the searches did not get stuck in the first selected action at each node. For an initial $W/N$ of $+\infty$, Hsueh et al. [32] reported similar results on NoGo, a deterministic game. Their experiments also suggested that extremely low $c_{puct}$ still learned well when the initial $W/N$ was $+\infty$. Since Section VIII continues

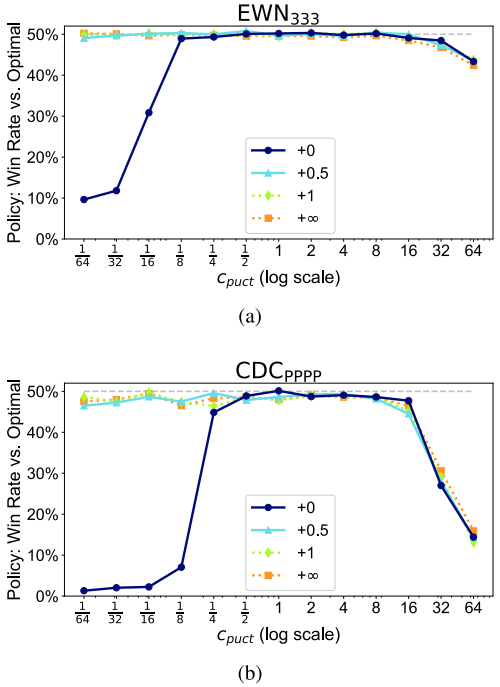



**FIGURE 23.** Last $WR_p$ of different initial $W/N$ for (a) $EWN_{333}$ and (b) $CDC_{PPPP}$ with different $c_{puct}$.

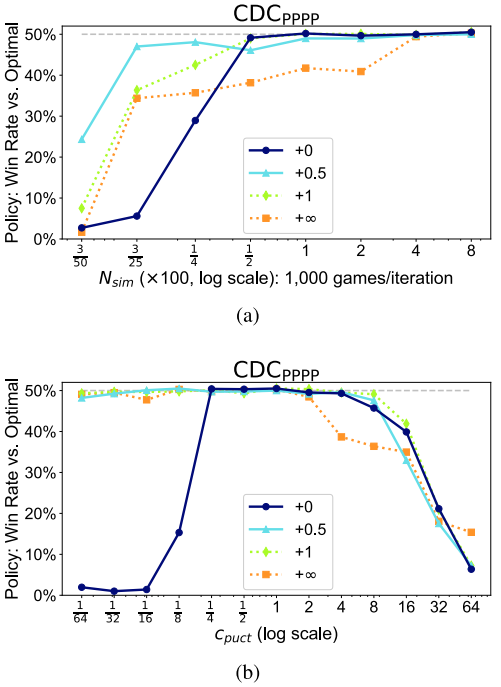



**FIGURE 24.** Last $WR_p$ of different initial $W/N$ for $CDC_{PPPP}$ with different (a) $N_{sim}$ and (b) $c_{puct}$ trained using the CLAP framework.

to experiment on initial $W/N$, an overall discussion on promising settings of initial $W/N$ will be made at the end of that section.

## VIII. EXPERIMENTS WITH NEURAL NETWORKS

In order to investigate whether there are gaps between the learning based on lookup tables and DNNs, some preliminary experiments on DNNs were conducted. In more detail, the CLAP framework [12] was employed, which was slightly extended to support stochastic games. CLAP implemented an asynchronous version of the AlphaZero algorithm, but the MCTS in self-play games was single-threaded and did not use the virtual loss technique discussed in Subsection VII-A.

In the experiments, the settings were similar to Section VII (i.e., different initial $W/N$ with a wide range of $N_{sim}$ or $c_{puct}$) and to Subsection V-D (i.e., different Dirichlet $\varepsilon$), and the game variant $CDC_{PPPP}$ was selected. Each setting was trained for 50 iterations on a server equipped with Intel (R) Xeon (R) Silver 4210R CPU @ 2.40GHz and NVIDIA Quadro RTX 8000 GPU. The training used all 40 CPU threads and the GPU on the server, taking about 5 hours for settings with an $N_{sim}$ of 800. The DNN structure and some more detailed settings are presented in Appendix B.

Fig. 24 shows the results of last $WR_p$ for settings similar to Section VII. Compared to Figs. 20b and 23b, the tendencies were generally similar. A minor difference was that an initial $W/N$ of $+\infty$ obtained slightly worse results (i.e., $N_{sim}$ between 25 to 200 and $c_{puct}$ of 4 and 8). As for Dirichlet $\varepsilon$, it was the same that the last $WR_p$ was close to 50% for $\varepsilon \geq 0.15$. Several trials for $\varepsilon = 0$ and 0.1 were run, where some had the last $WR_p$ close to 50% and some a bit lower (e.g., 42.0%), showing that the learning was unstable. To sum up, from the $WR_p$ results of $CDC_{PPPP}$ with different initial $W/N$, $c_{puct}$, $N_{sim}$, and Dirichlet $\varepsilon$, analyses based on lookup tables were generally applicable to DNNs.

From the experiments on both lookup tables and DNNs and on both $N_{sim}$ and $c_{puct}$ (under the case of single-threaded MCTS), an initial $W/N$ of $+0.5$ seemed the most promising, especially when $N_{sim}$ was extremely low (note again that high or low depends on factors such as the number of legal actions). As for why an initial $W/N$ of $+0.5$ could learn well with extremely low $N_{sim}$ but others could not, the reasons were considered to be related to randomly initialized policies and values, as discussed in the 4th and 5th paragraphs in Subsection VII-A. One thing to note for $+0.5$ is that it may have the problem of trying all actions exhaustively in disadvantageous positions (i.e., all actions leading to positions whose estimated values $<0.5$). It is expected that the problem will have more negative influences on games with more legal actions. For low $N_{sim}$ settings, a potentially promising idea is to pre-train lookup tables or DNNs with an initial $W/N$ of $+0.5$ and then switch to $+0$ for the rest of the learning. As for multi-threaded MCTS with virtual losses, $+0$ seemed the best choice, as discussed in the last paragraph in Subsection VII-A.

## IX. CONCLUSION AND FUTURE WORK

Regarding the milestone algorithm AlphaZero that mastered deterministic games such as chess, shogi, and Go without game-specific knowledge, this paper thoroughly investigated the algorithm on strongly-solved stochastic games: three

variants of EinStein würfelt nicht! (EWN) and three variants of $2 \times 4$ Chinese dark chess (CDC) on different scales. Different from deterministic games, positions' theoretical values are expected win rates in stochastic games instead of simply wins, losses, and draws. In addition, the search trees for stochastic games involve chance nodes. In order to distinguish good actions or positions from bad ones, it may be more important for stochastic games to learn values close to theoretical ones than deterministic games. Since strongly solved variants of games were employed, the experiments could compare the learning results with the optimal play and the theoretical values. Lookup tables were employed at first so that it would be easier to figure out why the learning succeeded or failed and less computationally costly to evaluate the learning results.

The experiments thoroughly studied hyper-parameters and some implementation details in the Monte-Carlo tree search (MCTS) for self-play games to see how the learning of the AlphaZero algorithm is influenced. The first was investigating the three hyper-parameters related to the action selection of the PUCT algorithm (1), including $c_{puct}$ and Dirichlet noise's $\alpha$ and $\varepsilon$. Policies learned by many tested settings played almost optimally against the optimal player, where some settings also learned values close to theoretical ones in opening positions. For hyper-parameters leading to higher levels of exploration in the PUCT algorithm, the ability to explore the state spaces was also higher. Second, the number of simulations per turn ($N_{sim}$) was investigated. The results suggested that a moderate number of simulations were sufficient for learning near-optimal actions. Although MCTS are usually stronger with higher $N_{sim}$, too-high $N_{sim}$ was unnecessary for the MCTS in self-play games. Third, different implementations of tree node initialization for MCTS were investigated. The experiments showed that different implementations made differences when $c_{puct}$ or $N_{sim}$ values were low, though the differences were limited in other cases. Finally, a preliminary experiment on a variant of $2 \times 4$ CDC ($\text{CDC}_{\text{PPPP}}$) was conducted, where lookup tables were replaced with deep neural networks (DNNs). The results showed that learning using lookup tables and using DNNs had similar tendencies, suggesting that the analyses through lookup tables could be generalized to DNNs.

In addition to showing promising ranges of hyper-parameters, another contribution of this paper was a thorough investigation of the reasons why the learning failed under some hyper-parameter settings. This could be done because strongly-solved game variants and lookup tables were employed so that each position could be checked individually and the results could be compared with the optimal play and the theoretical values. For example, when initializing tree nodes' $W/N$ to losses (+0), a low $N_{sim}$ (e.g., 6 for $\text{EWN}_{333}$) failed to learn the optimal play and theoretical values. The reasons were related to the random initialization of lookup tables and the PUCT algorithm. In the first iteration of learning, an action's $W/N$ was updated to $v \approx 0.5$ from the lookup table when it was selected during MCTS. For other actions, the $W/N$ was still the initialized +0, making them difficult to be selected within a few simulations by the PUCT algorithm. Thus, the learning was wrongly biased by randomness. When the initial $W/N$ was +0.5 (draws), such biases were greatly reduced. Although the investigation was done on small-scaled games with lookup tables, the phenomenon was expected to occur in larger-scaled games with DNNs since DNNs are also usually randomly initialized and the PUCT algorithm behaves in the same way.

The following discusses insights and suggestions to select or tune hyper-parameters obtained from the experiments in this paper. Considering the required computational resources and the learning results, it is practical to start by trying relatively low $N_{sim}$ such as 50 or 100, though games that have higher complexity are expected to require higher $N_{sim}$. When using single-threaded MCTS, this paper suggests that initializing tree nodes' $W/N$ to draws (+0.5) instead of losses (+0) can further lower the minimum required $N_{sim}$ to learn well. When using multi-thread MCTS with virtual losses, following Silver et al. [2] to initialize $W/N$ to losses (+0) is considered better. As for hyper-parameters related to the PUCT algorithm, the experiments have shown that a wide range exists for each hyper-parameter. For $c_{puct}$, the range of promising settings seems to decrease as games become complex, and settings around 1 to 2 can be tried first. For Dirichlet $\alpha$, when the average number of legal actions is relatively low, the experiments have shown that different $\alpha$ settings do not influence too much. How $\alpha$ influences learning in games with more legal actions remains an open question. For Dirichlet $\varepsilon$, 0.25, the setting used by Silver et al. [2], is a promising one to start with. The experiments have shown that the range of $[0.15, 0.5]$ does not differ too much and that $\varepsilon$ too low may make the learning unstable.

For future work, promising directions are discussed as follows. The first is to use DNNs to learn more complex games, such as the standard versions of EWN and CDC, and analyze how good the learned policies and values are. Although the standard versions have not been strongly-solved, endgame tables [21], [33] can be employed to investigate positions in endgames. Also, it is worth investigating hyper-parameters by approaches such as Bayesian optimization [3] and population based training [34], [35] other than grid search. In addition, it is worth investigating successors of the AlphaZero algorithm, such as MuZero [14], Stochastic MuZero [15], and Gumbel AlphaZero/MuZero [36].

## APPENDIX A
## ADDITIONAL EXPERIMENT RESULTS

This appendix presents additional experiment results, where Appendix A-A shows the results of multiple trials with the default setting and Appendix A-B shows the results of $\text{EWN}_{334}$, $\text{EWN}_{343}$, $\text{CDC}_{\text{KPPP}}$, and $\text{CDC}_{\text{GGCC}}$ that were not included in Sections V to VII.

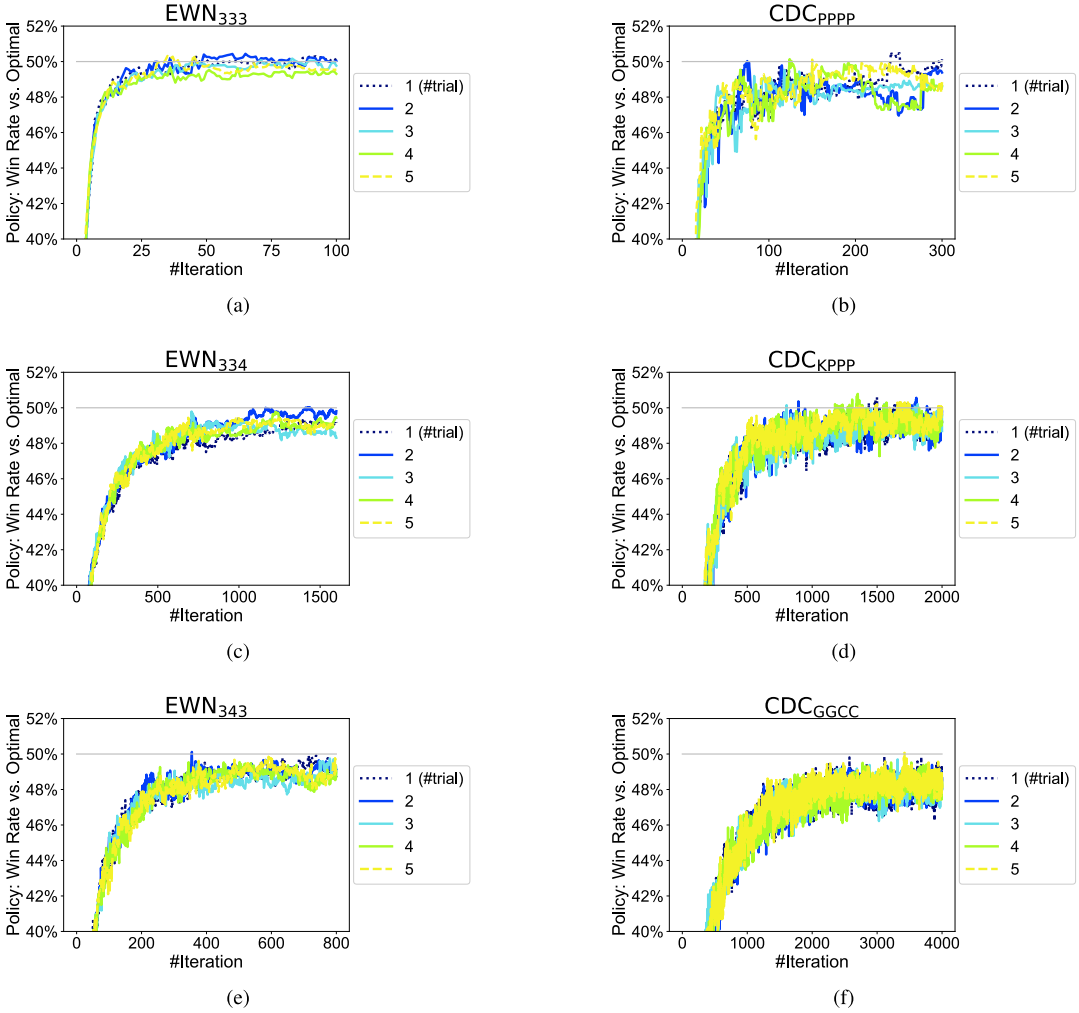



**FIGURE 25.** $WR_p$ training curves of different trials for the six selected game variants (left: EWN, right: 2 × 4 CDC).

### A. RESULTS OF MULTIPLE TRIALS

Fig. 25 shows the six game variants' $WR_p$ training curves of running the default setting five times with different random seeds. During learning, randomness was involved in the games themselves, the initialization of lookup tables, the Dirichlet noise, and the selection for actions to be played in self-play games (probabilities proportional to visit counts). The differences were very limited between different trials, demonstrating the repeatability of the tabular AlphaZero in stochastic games.

### B. SUPPLEMENTARY RESULTS AND DISCUSSIONS

Fig. 26 corresponds to Fig. 9, showing similar tendencies that too low and too high $c_{puct}$ both resulted in higher $ERR_p$ and $MAE_v$. Fig. 27 corresponds to Fig. 10, plotting the numbers of positions updated $\geq$1, $\geq$4, and $\geq$16 times with different $c_{puct}$ settings. EWN variants and 2 × 4 CDC variants themselves had similar tendencies, while the two groups of plots looked different. Since the game rules of EWN are different from 2 × 4 CDC, the game trees cover the state spaces in different ways. Thus, it is reasonable that the tendencies were not well-matched. Still, when $c_{puct}$ was in a moderate range (say 1/8 to 8), the number of updated positions increased as $c_{puct}$ increased as expected. Two exceptions were observed, EWN with too-low $c_{puct}$ and 2 × 4 CDC with too-high $c_{puct}$, discussed as follows.

For EWN, one of the victory conditions is to move a piece to the opponent's corner. For most positions in the experimented board sizes (3 × 3 and 3 × 4), the diagonal moves are the optimal actions if available, and moving pieces diagonally usually leads to shorter games. With too-low $c_{puct}$, the tabular AlphaZero might be wrongly biased by randomness and failed to learn the optimal actions. The self-play games were relatively long since diagonal moves were seldom played. Longer games were the major reason for having more updated positions for these extremely low $c_{puct}$ settings.

As for 2 × 4 CDC, too-high $c_{puct}$ made the action selection of MCTS in self-play games close to random. Playing randomly in 2 × 4 CDC increased the probability of just repeating the same positions and finally ending as draws due to the repetition rule. Within the same number of games,

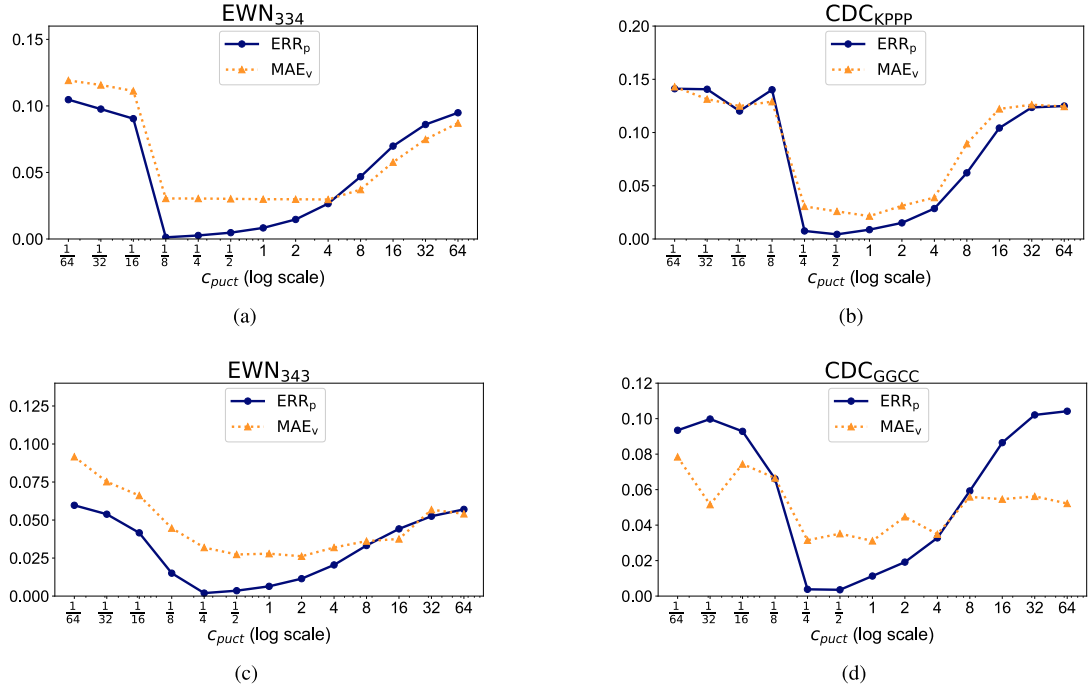



**FIGURE 26.** $ERR_p$ and $MAE_v$ of the last iterations with different $c_{puct}$ for (a) $EWN_{334}$, (b) $CDC_{KPPP}$, (c) $EWN_{343}$, and (d) $CDC_{GGCC}$.

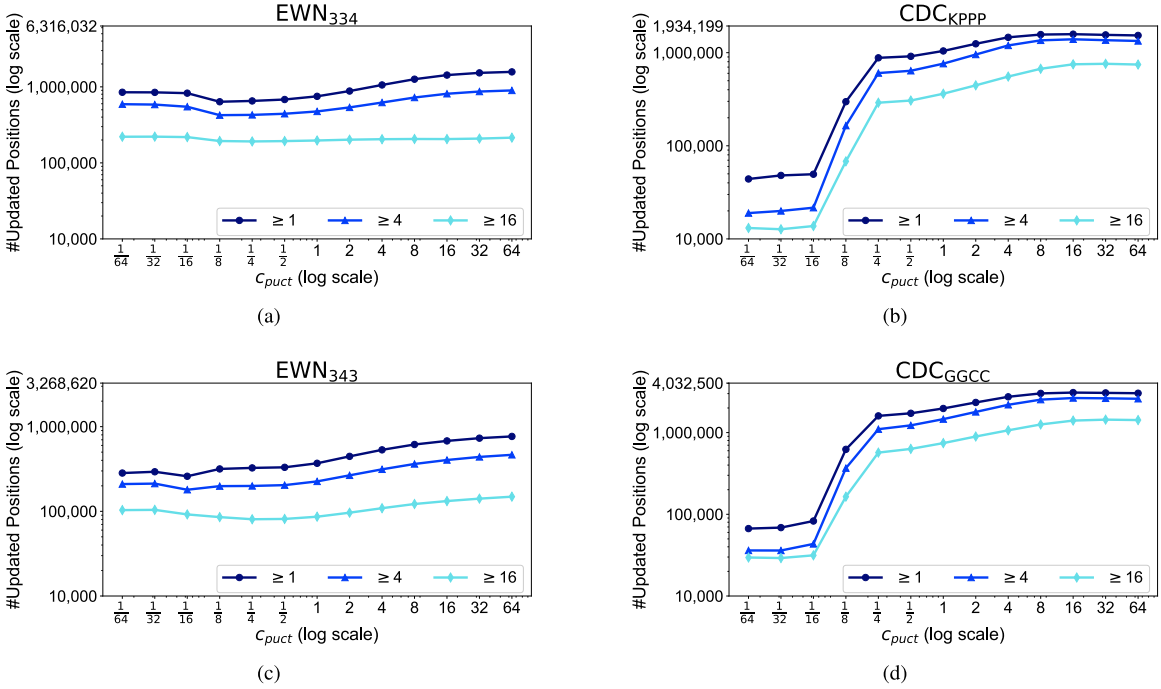



**FIGURE 27.** The number of positions updated ≥1, ≥4, and ≥16 times during training for (a) $EWN_{334}$, (b) $CDC_{KPPP}$, (c) $EWN_{343}$, and (d) $CDC_{GGCC}$ with different $c_{puct}$.

it was suspected that the diversity increased by exploring the state space was somewhat canceled by the increasing number of draw games due to repetition.

Fig. 28 corresponds to Fig. 12, depicting the $ERR_p$ and $MAE_v$ of the last iterations with different Dirichlet $\varepsilon$ settings. As discussed in the 2nd paragraph of Subsection V-D, settings failed to learn well (an $\varepsilon$ of 0 for $2 \times 4$ CDC variants particularly) had high $ERR_p$ and $MAE_v$, while the rest had relatively low $ERR_p$ and $MAE_v$ compared to bad settings of $c_{puct}$ (Figs. 9 and 26). Fig. 29 corresponds to Fig. 13, where the tendencies were similar that higher $\varepsilon$ led to more updated positions as expected.

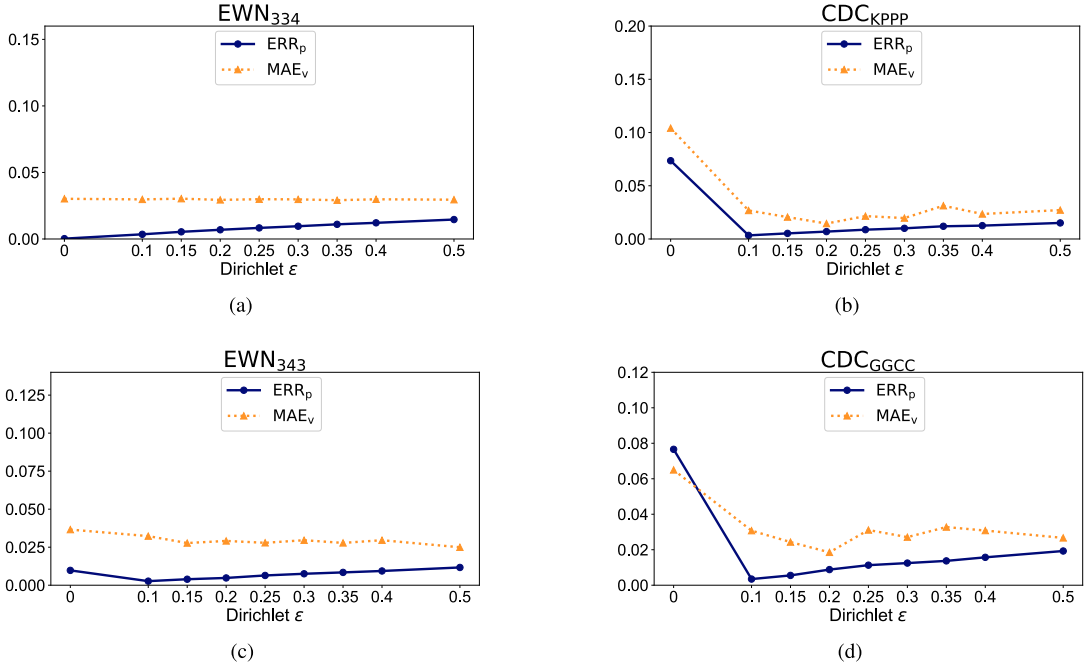



**FIGURE 28.** $ERR_p$ and $MAE_v$ of the last iterations with different $\varepsilon$ for (a) $EWN_{334}$, (b) $CDC_{KPPP}$, (c) $EWN_{343}$, and (d) $CDC_{GGCC}$.

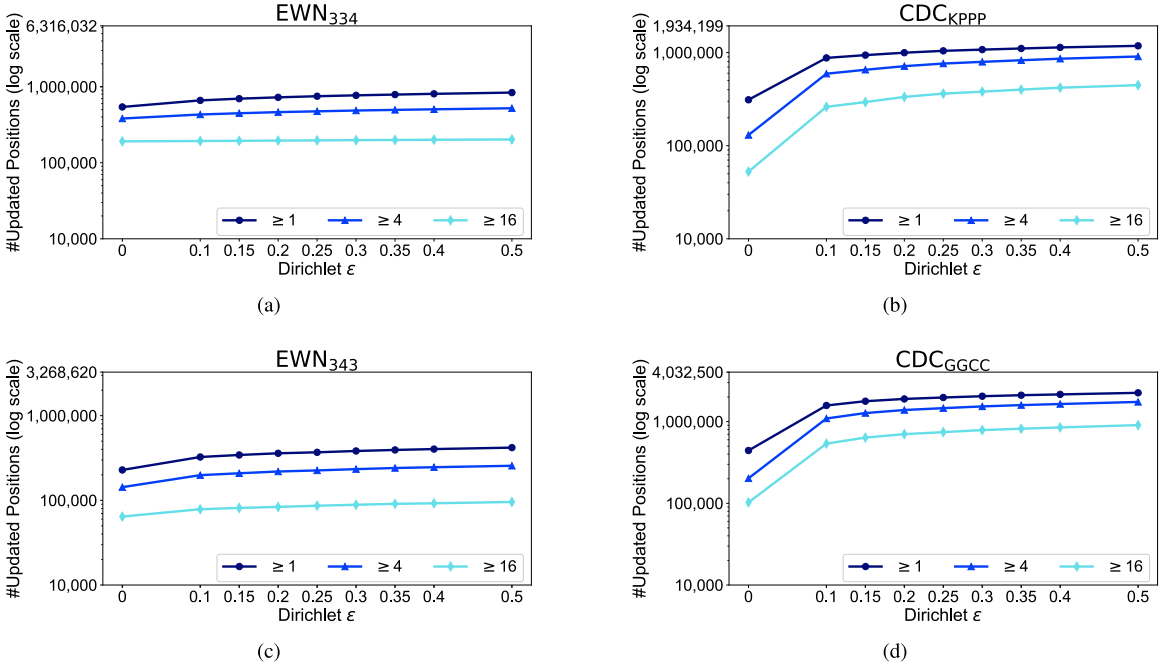



**FIGURE 29.** The number of positions updated ≥1, ≥4, and ≥16 times during training for (a) $EWN_{334}$, (b) $CDC_{KPPP}$, (c) $EWN_{343}$, and (d) $CDC_{GGCC}$ with different $\varepsilon$.

Figs. 30 and 31 correspond to Figs. 14 and 15, respectively, showing the $WR_p$ results with different $N_{sim}$ settings (1,000 self-play games/iteration). In all game variants, extremely low $N_{sim}$ (say 6 and 12) could not learn the optimal play. When $N_{sim}$ was sufficiently high (say 100), $WR_p$ was not significantly improved as $N_{sim}$ increased, where the $WR_p$ was close to 50%. From the results, a tendency was also observed that the minimal $N_{sim}$ to obtain $WR_p$ close to 50% was higher in more complex games in terms of the number of legal actions and game length. As a future research direction, it is interesting to investigate whether there are some relations between the minimal required $N_{sim}$ of a game and the game's state-space complexity, game-tree complexity, etc.[14]

[14]Danihelka et al. [36] proposed Gumbel AlphaZero/MuZero, able to learn well even with low $N_{sim}$, which is also worth investigating.

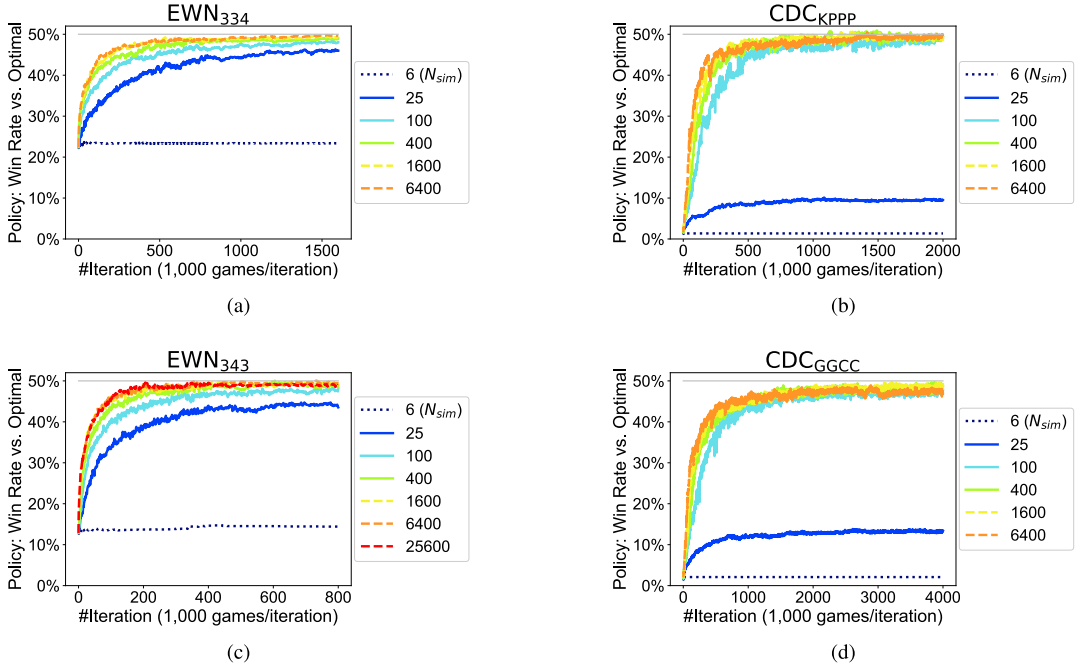



**FIGURE 30.** $WR_p$ training curves with different $N_{sim}$ under 1,000 games/iteration for (a) $EWN_{334}$, (b) $CDC_{KPPP}$, (c) $EWN_{343}$, and (d) $CDC_{GGCC}$.

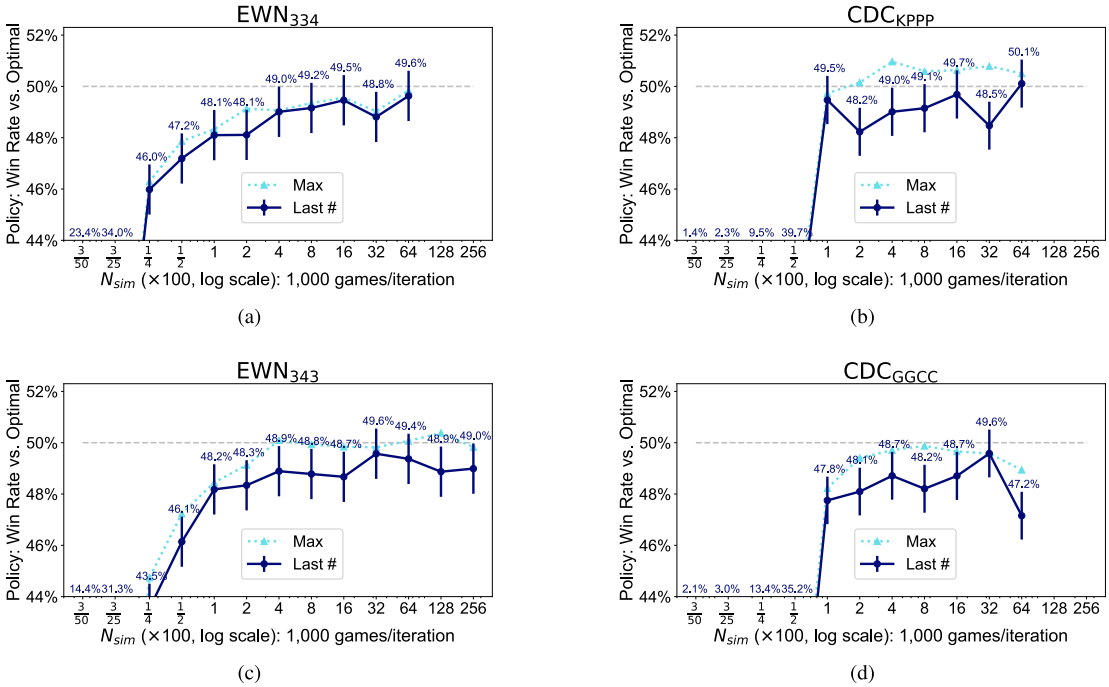



**FIGURE 31.** Last and max $WR_p$ with different $N_{sim}$ for (a) $EWN_{334}$, (b) $CDC_{KPPP}$, (c) $EWN_{343}$, and (d) $CDC_{GGCC}$ under 1,000 games/iteration.

Fig. 32 corresponds to Fig. 18, showing the $ERR_p$ and $MAE_v$ with different $N_{sim}$ settings under 1,000 self-play games per iteration. The tendencies were the same in that when $N_{sim}$ was sufficiently high to learn the optimal play, the $ERR_p$ and $MAE_v$ were relatively low compared to bad settings. Particularly, $ERR_p$ went close to 0 as $N_{sim}$ increased. Fig. 33 corresponds to Fig. 19, showing similar tendencies where the number of updated positions decreased as $N_{sim}$ increased for $N_{sim}$ settings that were sufficiently high to learn the optimal play (as discussed in the 6th paragraph of Subsection VI-B).

Fig. 34 corresponds to Fig. 20, presenting the $WR_p$ of the last iterations with different initial $W/N$ and $N_{sim}$ settings. Different game variants had similar tendencies: (i) when

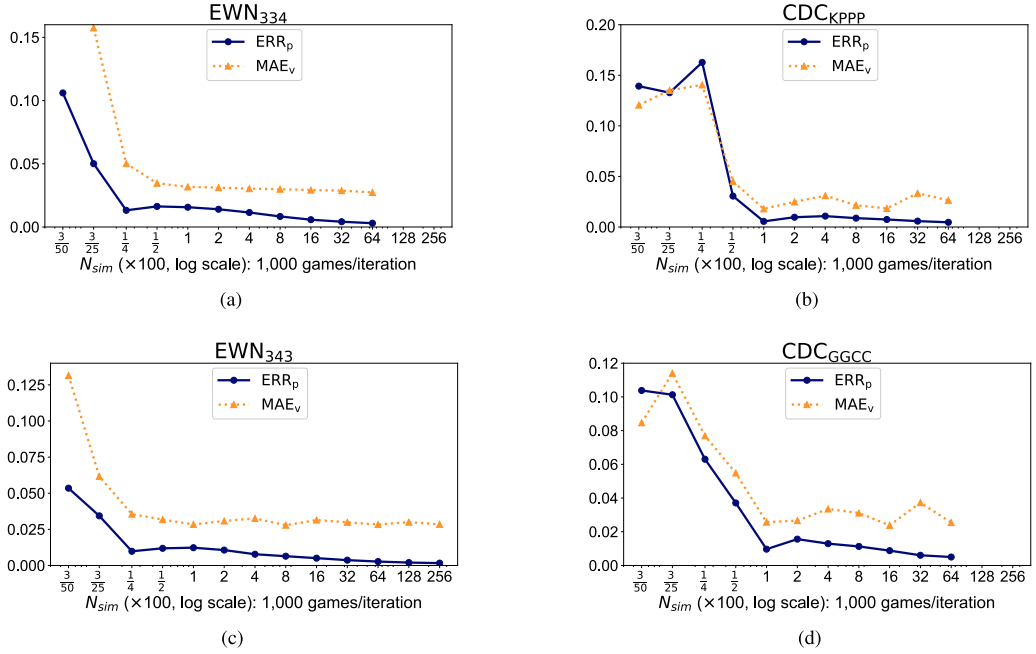



**FIGURE 32.** $\mathrm{ERR_p}$ and $\mathrm{MAE_v}$ of the last iterations with different $N_{sim}$ under 1,000 games/iteration for (a) $\mathrm{EWN_{334}}$, (b) $\mathrm{CDC_{KPPP}}$, (c) $\mathrm{EWN_{343}}$, and (d) $\mathrm{CDC_{GGCC}}$.

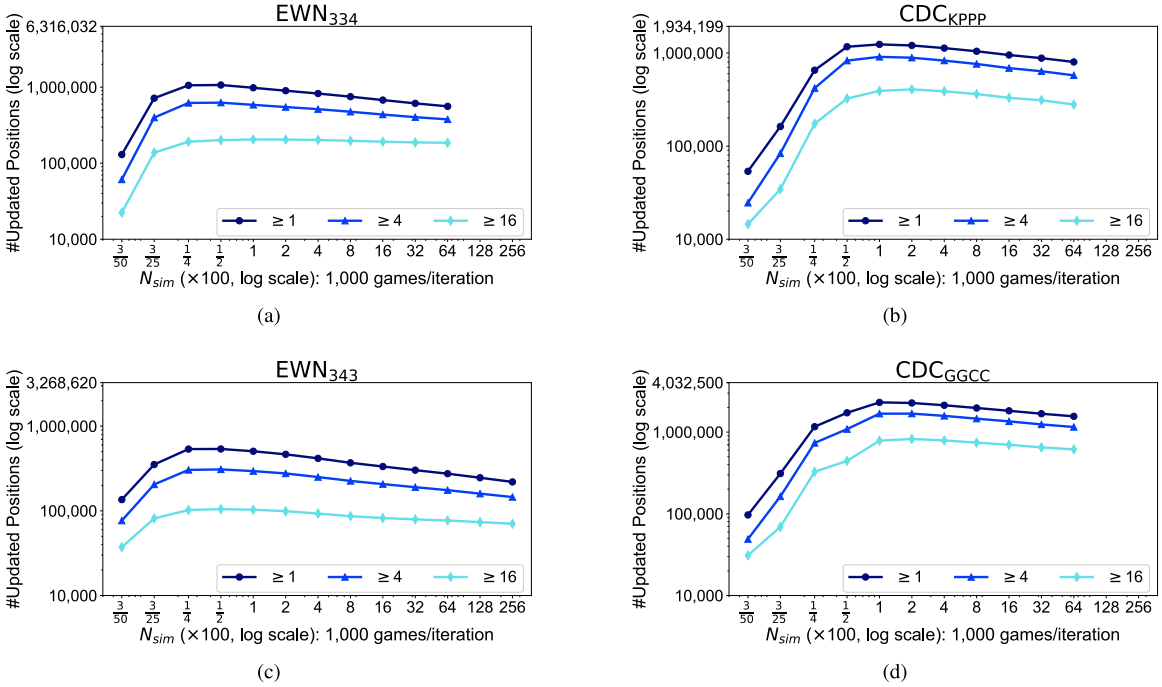



**FIGURE 33.** The number of positions updated ≥1, ≥4, and ≥16 times during training for (a) $\mathrm{EWN_{334}}$, (b) $\mathrm{CDC_{KPPP}}$, (c) $\mathrm{EWN_{343}}$, and (d) $\mathrm{CDC_{GGCC}}$ with different $N_{sim}$ under 1,000 self-play games/iteration.

$N_{sim}$ was sufficiently high, different initial $W/N$ did not differ too much in terms of the final $\mathrm{WR_p}$, and (ii) when $N_{sim}$ was extremely low, an initial $W/N$ of $+0.5$ worked the best, $+1$ and $+\infty$ got similar results, and $+0$ was the worst, as discussed in Subsection VII-A. Fig. 35 corresponds to Fig. 23, showing similar results that a great difference between an initial $W/N$ of $+0$ and the others were obtained when $c_{puct}$ was extremely low.

## APPENDIX B
## CLAP EXPERIMENT SETTINGS

This appendix presents detailed settings of the experiments employing the CLAP framework in Section VIII. The inputs to the DNNs consisted of 33 planes of size $2 \times 4$, as listed in Table 2. Features A–C represented the pieces on the board by a one-hot encoding using 15 binary planes. Features D and E counted the numbers of unrevealed pieces for

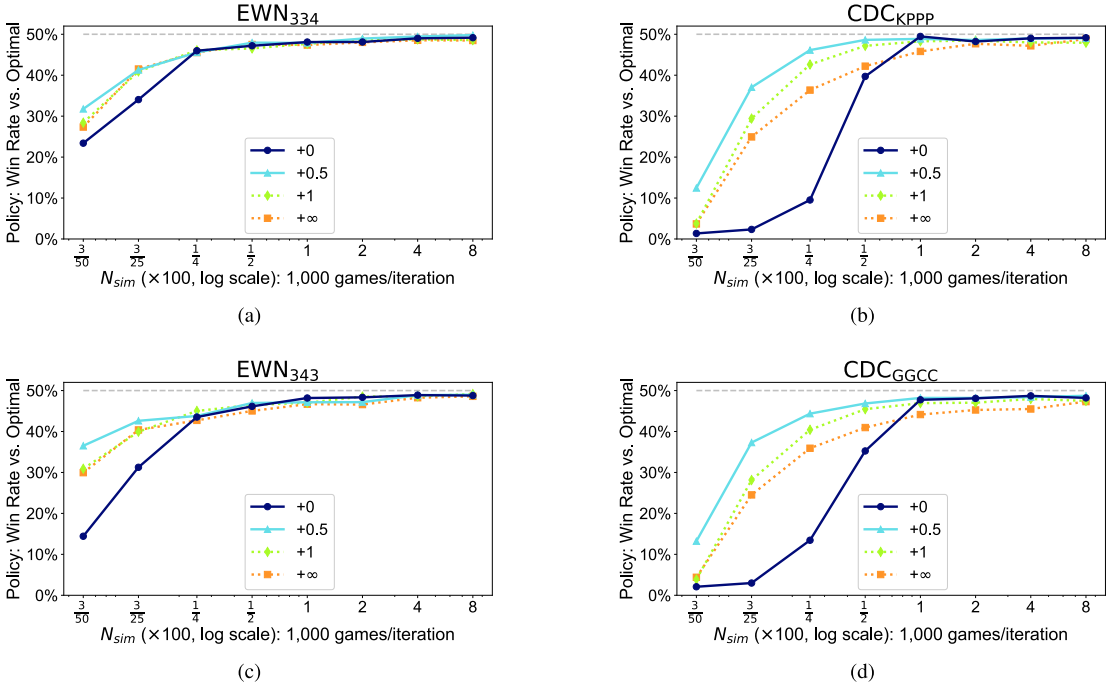



**FIGURE 34.** Last $WR_P$ of different initial $W/N$ for (a) $EWN_{334}$, (b) $CDC_{KPPP}$, (c) $EWN_{343}$, and (d) $CDC_{GGCC}$ with different $N_{sim}$.

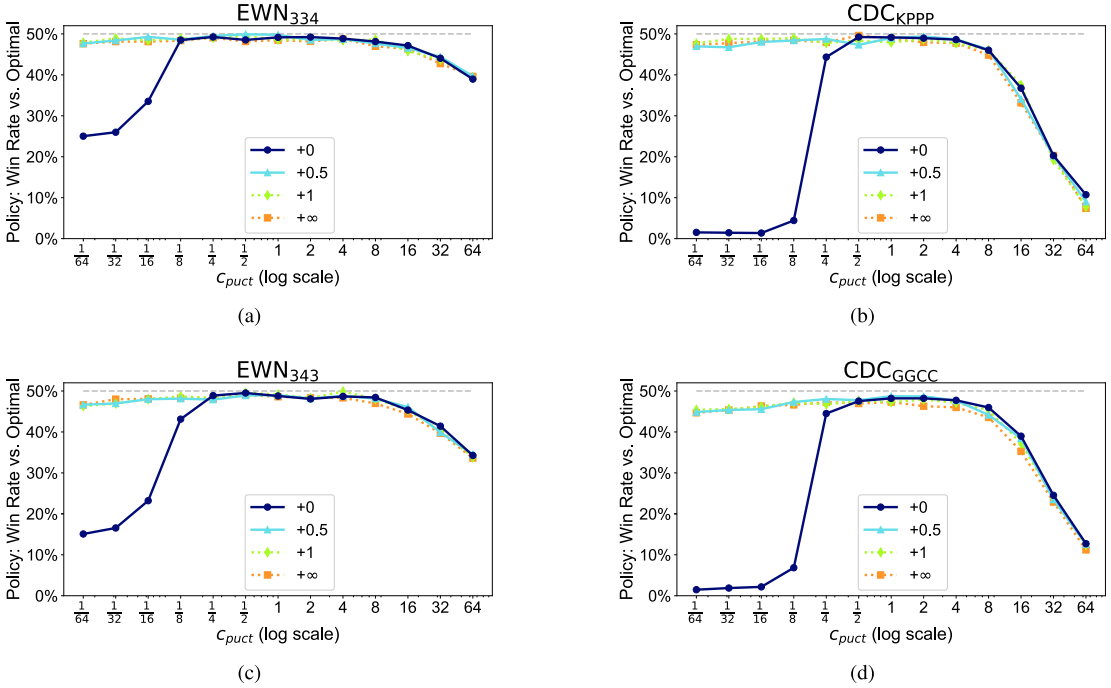



**FIGURE 35.** Last $WR_P$ of different initial $W/N$ for (a) $EWN_{334}$, (b) $CDC_{KPPP}$, (c) $EWN_{343}$, and (d) $CDC_{GGCC}$ with different $c_{puct}$.

each type and each player, filling the corresponding plane by repeating the same integer. Feature F represented the repetition count by 2 binary planes, and feature G represented by a binary plane whether it is the turn of the first player or the second player, similar to Silver et al. [2]'s design. Feature H represented the ratio of actions without flipping and capturing to the number leading to a draw game, filling the plane by repeating the same real number value. The inputs were then followed by a 5-block ResNet [37]. The outputs contained two heads, policy and value. The policy output was a 40-dimensional vector, the same as that for lookup tables explained in Section III. The value output was a 2-dimensional vector, estimating the two players' expected outcomes in $[-1, 1]$.

**TABLE 2. The input planes of DNNs for CDC.**

| Feature | #Planes | Feature | #Planes |
|---|---|---|---|
| (A) P1 piece | 7 | (F) Repetition | 2 |
| (B) P2 piece | 7 | (G) Player turn | 1 |
| (C) Faced-down piece | 7 | (H) No-progress rate | 1 |
| (D) P1 unrevealed count | 7 | | |
| (E) P2 unrevealed count | 7 | | |

The training for $\text{CDC}_{\text{PPPP}}$ lasted for 50 iterations. In each iteration, 1,000 self-play games were played, which was the same as the experiments on lookup tables. In the tabular AlphaZero (Algorithm 1), the 1,000 games were played sequentially one by one, while in CLAP, several games were played in parallel at the same time. The optimization mechanism was slightly different. CLAP had a replay buffer of size 40,000 to keep the latest games and updated the DNN by sampling batches of size 256 from the replay buffer. As a comparison, the tabular AlphaZero could be imagined to have a flexible size of replay buffer and a batch size of 1. The optimization algorithm in CLAP was based on stochastic gradient descent with momentum (0.9), weight decay (0.0001), and Nesterov's accelerated gradient [38]. The initial learning rate was 0.1 and was changed to 0.01 at the 25th iteration.

## REFERENCES


[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. P. 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, pp. 354–359, Oct. 2017.

[2] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018.

[3] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2015.

[4] B.-N. Chen, B.-J. Shen, and T.-S. Hsu, "Chinese dark chess," *ICGA J.*, vol. 33, no. 2, pp. 93–106, Jun. 2010.

[5] I. Althöfer. (2011). *On the Origins of 'EinStein Würfelt Nichtt!'*. Accessed: Apr. 7, 2022. [Online]. Available: https://althofer.de/origins-of-ewn.html

[6] H.-J. Chang and T.-S. Hsu, "A quantitative study of 2×4 Chinese dark chess," in *Proc. Int. Conf. Comput. Games*, Aug. 2013, pp. 151–162.

[7] F. Bonnet and S. Viennot, "Toward solving 'EinStein würfelt nicht!'" in *Proc. Adv. Comput. Games*, Jul. 2017, pp. 13–25.

[8] C.-H. Hsueh, I.-C. Wu, J.-C. Chen, and T.-S. Hsu, "AlphaZero for a non-deterministic game," in *Proc. Conf. Technol. Appl. Artif. Intell. (TAAI)*, Nov. 2018, pp. 116–121.

[9] Y. Tian, J. Ma, Q. Gong, S. Sengupta, Z. Chen, J. Pinkerton, and L. Zitnick, "ELF OpenGo: An analysis and open reimplementation of AlphaZero," in *Proc. Int. Conf. Mach. Learn. (ICML)*, vol. 97, Jun. 2019, pp. 6244–6253.

[10] L. Zero. (2017). *GitHub—Leela-Zero/Leela-Zero: Go Engine With no Human-Provided Knowledge, Modeled After the AlphaGo Zero Paper*. Accessed: Apr. 7, 2022. [Online]. Available: https://github.com/leela-zero/leela-zero

[11] T. Cazenave, Y. C. Chen, G. W. Chen, S. Y. Chen, X. D. Chiu, J. Dehos, M. Elsa, Q. Gong, H. Hu, V. Khalidov, and C. L. Li, "PolyGames: Improved zero learning," *ICGA J.*, vol. 42, no. 4, pp. 244–256, Jan. 2021.

[12] Y.-H. Chen, "A high-performance, distributed AlphaZero framework," M.S. thesis, Inst. Comput. Sci. Eng., Nat. Chiao Tung Univ., Hsinchu, Taiwan, 2020. [Online]. Available: https://hdl.handle.net/11296/64c8e8

[13] M. Lanctot, E. Lockhart, J. B. Lespiau, V. Zambaldi, S. Upadhyay, J. Perolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, and D. Hennes, "OpenSpiel: A framework for reinforcement learning in games," 2019, *arXiv:1908.09453*.

[14] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, and T. Lillicrap, "Mastering Atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[15] I. Antonoglou, J. Schrittwieser, S. Ozair, T. Hubert, and D. Silver, "Planning in stochastic environments with a learned model," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–30. [Online]. Available: https://openreview.net/forum?id=X6D9bAHhBQ1

[16] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set Monte Carlo tree search," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 120–143, Jun. 2012.

[17] C.-H. Hsueh, I.-C. Wu, W.-J. Tseng, S.-J. Yen, and J.-C. Chen, "An analysis for strength improvement of an MCTS-based program playing Chinese dark chess," *Theor. Comput. Sci.*, vol. 644, pp. 63–75, Sep. 2016.

[18] S.-J. Yen, S.-Y. Chiu, and I.-C. Wu, "MODARK wins the Chinese dark chess tournament," *ICGA J.*, vol. 33, no. 4, pp. 230–231, Dec. 2010.

[19] H. J. Van Den Herik, A. Plaat, and J. Hellemons, "The 16th computer olympiad," *ICGA J.*, vol. 35, no. 1, p. 50, 2012.

[20] H.-J. Chang, J.-C. Chen, C.-W. Hsueh, and T.-S. Hsu, "Analysis and efficient solutions for 2×4 Chinese dark chess," *ICGA J.*, vol. 40, no. 2, pp. 61–76, 2018.

[21] J.-C. Chen, T.-Y. Lin, B.-N. Chen, and T.-S. Hsu, "Equivalence classes in Chinese dark chess endgames," *IEEE Trans. Comput. Intell. AI Games*, vol. 7, no. 2, pp. 109–122, Jun. 2015.

[22] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. New York, NY, USA: Springer-Verlag, 2006.

[23] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *Proc. IWANN*, Jun. 1995, pp. 195–201.

[24] C.-H. Hsueh, "On strength analyses of computer programs for stochastic games with perfect information," Ph.D. dissertation, Inst. Comput. Sci. Eng., Nat. Chiao Tung Univ., Hsinchu, Taiwan, 2019. [Online]. Available: https://hdl.handle.net/11296/ku48z7

[25] C.-H. Hsueh, I.-C. Wu, T.-S. Hsu, and J.-C. Chen, "An investigation of strength analysis metrics for game-playing programs: A case study in Chinese dark chess," *ICGA J.*, vol. 40, no. 2, pp. 77–104, Feb. 2019.

[26] (Dec. 2018). *Alphazero News*. Accessed: Apr. 7, 2022. [Online]. Available: http://talkchess.com/forum3/viewtopic.php?t=69175&start=70#p781765

[27] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. Eur. Conf. Mach. Learn.*, Sep. 2006, pp. 282–293.

[28] C. B. Browne, E. J. 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.

[29] (Dec. 2018). *Alphazero News*. Accessed: Apr. 7, 2022. [Online]. Available: http://talkchess.com/forum3/viewtopic.php?t=69175&start=80#p781797

[30] 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.

[31] G. M. J. B. Chaslot, M. H. M. Winands, and H. J. Van Den Herik, "Parallel Monte-Carlo tree search," in *Proc. 6th Int. Conf. Comput. Games*. Berlin, Germany: Springer, 2008, pp. 60–71.

[32] C.-H. Hsueh, K. Ikeda, S.-G. Nam, and I.-C. Wu, "Analyses of tabular AlphaZero on NoGo," in *Proc. Int. Conf. Technol. Appl. Artif. Intell. (TAAI)*, Dec. 2020, pp. 254–259.

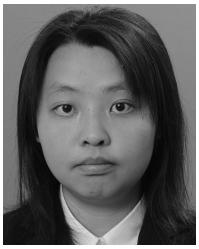
[33] W. Turner, "EinStein würfelt nicht—An analysis of endgame play," *ICGA J.*, vol. 35, no. 2, pp. 94–102, Jun. 2012.

[34] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population based training of neural networks," 2017, *arXiv:1711.09846*.

[35] T.-R. Wu, T.-H. Wei, and I.-C. Wu, "Accelerating and improving AlphaZero using population based training," in *Proc. AAAI Conf. Artif. Intell.*, Feb. 2020, pp. 1046–1053.

[36] I. Danihelka, A. Guez, J. Schrittwieser, and D. Silver, "Policy improvement by planning with Gumbel," in *Proc. 10th Int. Conf. Learn. Represent.*, 2022, pp. 1–22. [Online]. Available: https://openreview.net/forum?id=bERaNdoegnO

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.
[38] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. ICML*, vol. 28. Apr. 2013, pp. 1139–1147.



**CHU-HSUAN HSUEH** (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from the National Chiao Tung University, Hsinchu, Taiwan, in 2014 and 2019, respectively.

She is currently an Assistant Professor with the Japan Advanced Institute of Science and Technology, Nomi, Ishikawa, Japan. Her research interests include artificial intelligence, machine learning, and computer games.

**KOKOLO IKEDA** received the B.S. degree from The University of Tokyo, in 1999, the M.E. degree from the Tokyo Institute of Technology, in 2000, and the D.E. degree, in 2003.

He is currently a Professor with the Japan Advanced Institute of Science and Technology. His research interests include optimization and game informatics.

Dr. Ikeda is also the Vice Chairperson of the Computer Go Forum. He was one of the Program Chairs of the 2020 IEEE Conference on Games.

**I-CHEN WU** (Senior Member, IEEE) received the B.S. degree in electronic engineering and the M.S. degree in computer science from the National Taiwan University, Taipei, Taiwan, in 1982 and 1984, respectively, and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, USA, in 1993.

He is currently the Executive Officer of the Artificial Intelligence Computing Center, Academia Sinica, a Research Fellow of the Research Center for IT Innovation, Academia Sinica, and also a Professor with the Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan. His research achievements include several state-of-the-art game-playing programs, such as CGI for Go and Chimo for Chinese Chess, winning more than 30 gold medals in international tournaments, like Computer Olympiad. He wrote more than 150 technical papers and served as the chair and a member of committees for more than 30 academic conferences and organizations, including the Conference Chair of the 2015 IEEE CIG Conference. His research interests include computer games and deep reinforcement learning.

Dr. Wu is the Editor-in-Chief of *ICGA Journal* and an Associate Editor of the IEEE TRANSACTIONS ON GAMES. He was the President of the Taiwanese Association for Artificial Intelligence, from 2015 to 2017. He is also the Vice President of the International Computer Games Association and the President of the Taiwanese Computer Games Association.

**JR-CHANG CHEN** received the B.S., M.S., and Ph.D. degrees in computer science and information engineering from the National Taiwan University, Taipei, Taiwan, in 1996, 1998, and 2005, respectively.

He is currently a Professor with the Department of Computer Science and Information Engineering, National Taipei University. He is the coauthor of the two Chinese chess programs named *Elp* and *Chimo*, and the Chinese dark chess program named *Yahari*, which have won gold and silver medals in the Computer Olympiad tournaments. His research interests include artificial intelligence and computer games. From 2015 to 2016, he was the Secretary General of the Taiwanese Association for Artificial Intelligence.

**TSAN-SHENG HSU** (Senior Member, IEEE) received the B.S. degree in computer science and information engineering from the National Taiwan University, Taipei, Taiwan, and the M.S. and Ph.D. degrees in computer sciences from The University of Texas at Austin, Austin, TX, USA.

He has been a tenured full Research Fellow of the Institute of Information Science, Academia Sinica, Taipei, since June 2003. He has also been an Adjunct Professor with the Department of Computer Science, National Taiwan University, since 2003. His research interests include graph theory and its applications, computer games, data privacy, and design, analysis, implementation, and performance evaluation of algorithms.

Dr. Hsu is a member of the ACM and ACM SIGACT, a Senior Member of the IEEE Computer Society, and a Life Member of IICM.