AlphaZero for a Non-deterministic Game

Chu-Hsuan Hsueh and I-Chen Wu
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan
{hsuehch, icwu}@aigames.nctu.edu.tw

Jr-Chang Chen
Department of Computer Science and Information Engineering
National Taipei University
New Taipei City, Taiwan
jcchen@mail.ntpu.edu.tw

Tsan-sheng Hsu
Institute of Information Science
Academia Sinica
Taipei, Taiwan
tshsu@iis.sinica.edu.tw

***Abstract*—The AlphaZero algorithm, developed by DeepMind, achieved superhuman levels of play in the games of chess, shogi, and Go, by learning without domain-specific knowledge except game rules. This paper investigates whether the algorithm can also learn theoretical values and optimal plays for non-deterministic games. Since the theoretical values of such games are expected win rates, not a simple win, loss, or draw, it is worthy investigating the ability of the AlphaZero algorithm to approximate expected win rates of positions. This paper also studies how the algorithm is influenced by a set of hyper-parameters. The tested non-deterministic game is a reduced and solved version of Chinese dark chess (CDC), called 2×4 CDC. The experiments show that the AlphaZero algorithm converges nearly to the theoretical values and the optimal plays in many of the settings of the hyper-parameters. To our knowledge, this is the first research paper that applies the AlphaZero algorithm to non-deterministic games.**

Keywords-AlphaZero; non-deterministic game; Chinese dark chess; theoretical value

I. Introduction

In 2017, Silver *et al.* presented a program named AlphaGo Zero [1] 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. Later in the same year, they generalized the approach to the AlphaZero algorithm [2], and successfully applied it to chess and shogi as well as Go. The trained programs defeated world-champion programs in each of the three games.

Both AlphaGo Zero and the AlphaZero algorithm have some hyper-parameters. Silver *et al.* tuned hyper-parameters in AlphaGo Zero by Bayesian optimization [1, 3]. In the AlphaZero algorithm, most of the hyper-parameters followed those tuned in AlphaGo Zero [2]. However, they specifically tuned one of the hyper-parameters (the α in Dirichlet noise) for different games according to the approximate number of legal moves. It is interesting to investigate how the AlphaZero algorithm is influenced by hyper-parameters.

Moreover, it is also worthy studying whether the AlphaZero algorithm learns theoretical values and optimal plays even in non-deterministic games. The algorithm has been demonstrated to be successfully applied to 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. On the contrary, in non-deterministic games such as Chinese dark chess (CDC) [4-6], the theoretical values are expected win rates. It is interesting to investigate the ability of the AlphaZero algorithm to approximate expected win rates.

However, CDC, normally played on a 4×8 board with 32 pieces, is still too large to be solved. A reduced version, 2×4 CDC [7, 8], has been solved by Chang *et al.*, and is used in this paper to verify whether the AlphaZero algorithm approximates to the expected win rates. Instead of using neural networks to extract features, a lookup table keeps for each position the probabilities of moves and the estimated value, since the state space is small enough. Generally, the algorithm converged to nearly the theoretical values and the optimal plays of the game in many of the tested settings. This demonstrates the robustness of the AlphaZero algorithm.

The rest of the paper is structured as follows. Section II describes background knowledge including AlphaGo Zero and the AlphaZero algorithm, the game of CDC, and the 2×4 CDC. Section III then illustrates the tabular AlphaZero algorithm applied to 2×4 CDC, and the experiment results are included in Section IV. Finally, Section V makes concluding remarks and discusses further research directions.

II. Background

In this section, AlphaGo Zero and the AlphaZero algorithm are reviewed in Subsection II.A. The game of CDC and the reduced 2×4 CDC are described in Subsections II.B and II.C respectively.

A. AlphaGo Zero and the AlphaZero Algorithm

In AlphaGo Zero [1], deep neural networks (DNNs) were trained by reinforcement learning to predict policies (i.e., probability distribution of moves) and values given positions. The key idea was to make the AlphaGo 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 moves with higher quality, and then the DNNs were trained to predict the moves selected by tree search and the final winner in the self-play games.

In more detail, the learning algorithm contained three parts including self-play, optimization, and evaluation. These three parts were executed asynchronously in parallel in a large-scale computation system.

Self-play games were generated by Monte-Carlo tree search (MCTS) with 1,600 simulations per move. MCTS in AlphaGo Zero contained three phases in each simulation, which were selection, expansion, and backpropagation. In the selection phase, PUCT algorithm

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

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 constant influencing the level of exploration in search. During the expansion phase, the leaf node s_L was evaluated by DNN to obtain the value v and the policy $\boldsymbol{p}$, and then expanded with each of the actions initialized to $N(s_L,a)=0$, $W(s_L,a)=0$ and $P(s_L,a)=p_a$. Backpropagation updated the traversed nodes by $N(s,a)=N(s,a)+1$ and $W(s,a)=W(s,a)+v$. Finally, the move to be actually played at the root was chosen in proportion to the visit count. More specifically, the probability to select move a at the root s_0 was $\pi(a|s_0) = N(s_0,a)^{1/\tau}/\sum_b N(s_0,b)^{1/\tau}$ where τ is a temperature parameter determining the level of exploration in self-play games. At the end of a self-play game, the policies $\boldsymbol{\pi}$'s and the final outcome z of the game were saved as training data.

To ensure all legal moves at the root may be tried, Dirichlet noises were added to the prior probabilities in the root node. The prior probability $P(s_0,a)$ became $(1-\varepsilon)p_a + \varepsilon\eta_a$ where η_a is sampled from Dirichlet distribution $Dir(\alpha)$ with α controlling the level of concentration, and ε is the weight of the noise. Under the same numbers of samples, lower α's generally lead to more concentrated distribution, and thus higher levels of exploration.

In optimization, AlphaGo Zero randomly sampled positions from the most recent 500,000 self-play games. The DNNs were then optimized by stochastic gradient descent using the positions and the corresponding policies and values. The optimized DNN was evaluated by playing games against the current *best* DNN. Both of the DNNs were incorporated into MCTS with 1,600 simulations per move. If the new DNN obtained a win rate greater than 55% in 400 games, it became the new best DNN. The self-play part always used the best DNNs to generate games, and the first best DNN was the one with weights randomly initialized. The evaluation part was to ensure that the quality of the collected self-play games either remained the same or became better. The resulting program achieved superhuman performance, defeating the AlphaGo program that won against the world champion in 2016 [9].

The AlphaZero algorithm [2] was generalized from AlphaGo Zero. First, in addition to binary outcomes of wins and losses, draws were taken into consideration. Second, the evaluation part was removed, which meant that only one DNN was maintained and updated continually. The algorithm was tested on three games which were chess, shogi, and Go, and achieved superhuman levels of play.

In AlphaGo Zero, the hyper-parameters related to MCTS were selected by Bayesian optimization. The AlphaZero algorithm reused the hyper-parameters for all the three games, except the α parameter in the Dirichlet noise. The value was scaled proportional to the approximate numbers of legal moves in a typical position of the games. More specifically, the values were 0.03 for Go, 0.15 for shogi, and 0.3 for chess.

The codes of AlphaGo Zero and AlphaZero were not released. The open-sourced projects including Leela Zero [10], Mingo [11], and ELF Open Go [12] try to reproduce the results of the algorithm in the game of Go.

B. Chinese Dark Chess

Chinese dark chess (CDC) [4-6] is a two-player zero-sum non-deterministic game widely played in Taiwan, and also a game played in Computer Olympiad since 2010 [13-19]. 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 then put in the squares on the board. The first player reveals one piece to make it faced-up, and owns the set of pieces with the same color as the revealed one. The second player automatically owns the other set. 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 adjacent squares either without pieces or with an opponent 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 opponent pieces with equal or lower ranks. However, kings cannot capture pawns while pawns can capture kings. 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 pieces in the same row or the same column with exactly one piece in between.

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

C. 2×4 Chinese Dark Chess

Chang *et al.* introduced and solved a reduced version of CDC, called 2×4 CDC [7, 8]. The board size was reduced to 2×4 and only eight pieces were used. The initial position is shown in Fig. 1 (a). In 2×4 CDC, 24 fair and equivalent material combinations (combinations of pieces) were considered, as listed in Table I. Fair material combinations are those that both players own the same set of pieces at the start of the games. For example, "`KGGM` vs. `kggm`" is fair while "`KGGM` vs. `kggr`" is not. As for equivalence [20], 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`" thus is not counted.

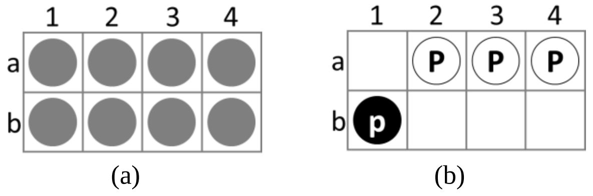

Fig. 1. (a) The initial position and (b) an example of position for 2×4 CDC.

TABLE I. 24 MATERIAL COMBINATIONS IN 2×4 CDC

`KGGM` vs. `kggm`	`KGGC` vs. `kggc`	`KGGP` vs. `kggp`
`KGMM` vs. `kgmm`	`KGMR` vs. `kgmr`	`KGMC` vs. `kgmc`
`KGMP` vs. `kgmp`	`KGCC` vs. `kgcc`	`KGCP` vs. `kgcp`
`KCCP` vs. `kccp`	`KCPP` vs. `kcpp`	`KPPP` vs. `kppp`
`GGMM` vs. `ggmm`	`GGMR` vs. `ggmr`	`GGMC` vs. `ggmc`
`GGCC` vs. `ggcc`	`GGCP` vs. `ggcp`	`GMCP` vs. `gmcp`
`GCCP` vs. `gccp`	`GCPP` vs. `gcpp`	`GPPP` vs. `gppp`
`CCPP` vs. `ccpp`	`CPPP` vs. `cppp`	`PPPP` vs. `pppp`

Chang *et al.* assigned theoretical values to the positions from retrograde analysis in the expectiminimax manner [7, 8]. 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 point 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. 1 (b), a position from a game of "PPPP vs. pppp." The result is a win for the red player. For a non-deterministic position, the game outcomes might vary according to different piece types revealed. Thus, the theoretical value was a real number between -1 and 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.

Hsueh *et al.* analyzed the strengths of game-playing programs on 2×4 CDC [6]. The results demonstrated that win rates against practically available programs such as MCTS and expectiminimax had high correlations to the win rates against the optimal play. In addition, they also showed that the win rates obtained in 2×4 CDC may be used to predict those in 4×8 CDC in some cases.

III. Tabular AlphaZero for 2×4 CDC

This paper replaces the DNNs in the AlphaZero algorithm by lookup tables since the state space is small enough. In a lookup table for 2×4 CDC, each position has its own policy and value. The policy is represented by a vector of real numbers $\hat{\boldsymbol{p}}$. The size of the vector is 40, which is the total number of all possible moves in 2×4 CDC (eight for flipping at a1, a2, …, b4 and the rest for moving a1→a2, a1→a3, a1→a4, a1→b1, a2→a1, …, b4→b3). The probability distribution of the moves is calculated by the softmax function $p_a = e^{(\hat{p}_a)}/\sum_b e^{(\hat{p}_b)}$ [21]. A real number $\hat{v}$ is used to represent the value of a position, and is scaled by the sigmoid function $v = 1/\left(1 + e^{-\hat{v}}\right)$ [22] to the range of $[0, 1]$.

From the collected self-play games, the updates for one position, with the corresponding policy $\boldsymbol{\pi}$ and the game outcome z, are performed as follows. For policy, the goal is to have the estimated probability distribution $\boldsymbol{p}$ (obtained from the lookup table) similar to $\boldsymbol{\pi}$. As for value, the goal is to have the estimated value v close to z. Following AlphaGo Zero, 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 [1]. Thus, policy and value in lookup tables are updated by $\hat{p}_a \leftarrow \hat{p}_a - lr \cdot (p_a - \pi_a)$ and $\hat{v} \leftarrow \hat{v} - lr \cdot (v - z) \cdot v \cdot (1 - v)$ respectively, where lr is the learning rate. The derivations are omitted due to page limit.

A synchronous version of the AlphaZero algorithm, as presented in Algorithm 1, is used to train the lookup tables for 2×4 CDC. At 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*×*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. The process of self-play and optimization repeats for *N* times in total.

Algorithm 1 – Tabular AlphaZero for 2×4 CDC

```
1:  Initialize the lookup table
2:  Repeat N iterations
3:      Collect M self-play games
4:      Optimize by the most recent K×M games (if any)
```

IV. Experiments

In this section, settings of experiments are illustrated in Subsection IV.A. The results on varying c_{puct}, Dirichlet α, Dirichlet ε, and temperature τ are then shown in Subsections IV.B, IV.C, IV.D, and IV.E respectively. Finally, Subsection IV.F gives overall discussions on the results.

A. Settings of Experiments

In the preliminary study, the experiments were conducted based on the material combination "PPPP vs. pppp." The total number of possible positions in the games starting from this combination is 194,933, and the theoretical value of the initial position is about 0.2286.

Four hyper-parameters, all related to exploration, c_{puct}, Dirichlet α, Dirichlet ε, and temperature τ were investigated. The default values were 1, 1.5, 0.25, and 1 respectively. In the experiments, the policies and values in the lookup table were initialized by a normal distribution with the mean of 0 and the variance of 0.01. The values of *M*, *N*, and *K* in Algorithm 1 were set to 1,000, 50, and 2 respectively. MCTS in self-play used 800 simulations per move, as the AlphaZero algorithm [2]. The learning rate lr was set to 1 at the beginning, and then decreased to 0.1 at the 25th iteration to stabilize the learning.

At the end of each iteration, the lookup table was evaluated by the following three metrics. The first was to play 10,000 games against the optimal play [6], with 5,000 games playing as the first player and the other 5,000 as the second. The lookup table agent played the move with the highest probability among the 40 moves. In cases that illegal moves were selected, the agent lost immediately. This metric aimed to evaluate the learned policy. The second was the absolute error (AE) between the estimated value and the theoretical value of the initial position, which was used to judge whether the algorithm was able to learn the theoretical value of the game. The third metric was the number of distinct positions seen so far, which was used to measure the level of exploration.

B. Results on c_{puct}

The tested values for c_{puct} included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16. The training curves, composed of the win rates obtained at the end of each iteration, are depicted in Fig. 2. With an extremely low level of exploration ($c_{puct} = 0.0625$), the AlphaZero algorithm failed to learn to play the game. For $c_{puct} \geq 1$, the speed of learning decreased as c_{puct} increased (higher levels of exploration). In addition, the progress of learning became unstable for larger values of c_{puct}.

The win rates achieved by different c_{puct}'s at the 50th iteration are shown in Fig. 3. The 95% confidence intervals are also shown, though they are not clearly visible (about 0.98%). Except for the two extreme ends $c_{puct} = 0.0625$ and $c_{puct} = 16$, the win rates were all close to 50%. Among these settings, $c_{puct} = 0.5$ (49.44±0.98%) and $c_{puct} = 1$ (49.60 ±0.98%) were considered to reach the optimal play.

Fig. 4 shows the AEs for different c_{puct}'s, averaged from the last five iterations. The 95% confidence intervals are also depicted in Fig. 4. The lowest AE was reached at $c_{puct} = 0.5$ (0.0131±0.0076). The settings of $c_{puct} = 0.125$, $c_{puct} = 0.25$, and $c_{puct} = 1$ also had low AEs. This demonstrated that the algorithm could approximate the theoretical value of the game well. For $c_{puct} \geq 1$, the AE increased significantly as c_{puct} increased.

The total numbers of distinct seen positions during training for different c_{puct}'s are depicted in Fig. 5. As expected, a higher c_{puct} (a higher level of exploration) led to a higher number of seen positions, with only one exception $c_{puct} = 16$.

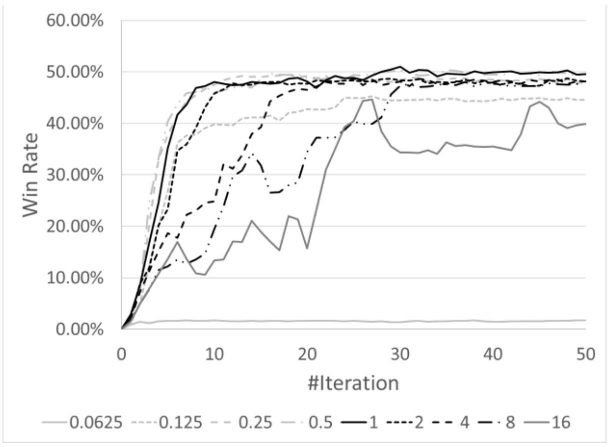

Fig. 2. Training curves for different c_{puct}.

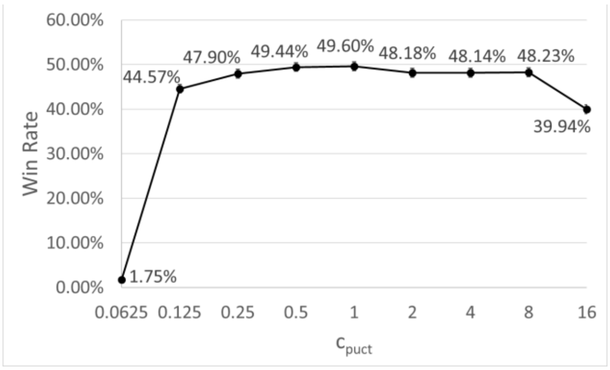

Fig. 3. The win rates at the 50th iteration for different c_{puct}.

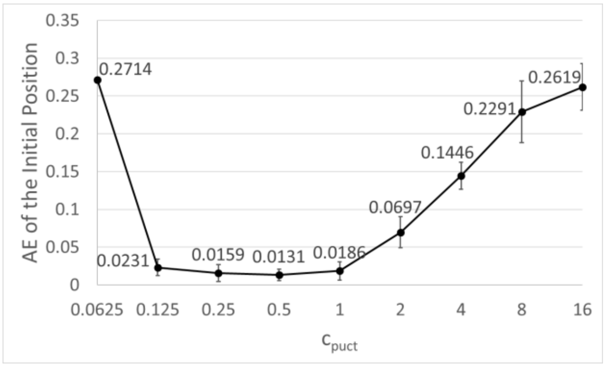

Fig. 4. The averaged AEs of the initial position over the last five iterations for different c_{puct}.

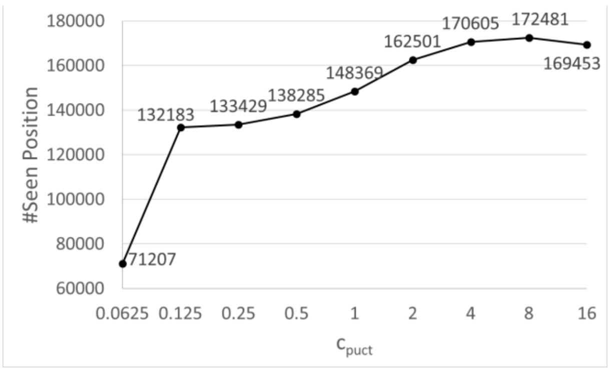

Fig. 5. The total numbers of distinct seen positions during training for different c_{puct}.

C. *Results on Dirichlet* α

The tested values of Dirichlet α included 0.03, 0.15, 0.3, 0.75, 1.5, 3, 6, 9, and 12. The training curves are not presented since they did not have much difference. Fig. 6 shows the win rates achieved by different Dirichlet α's at the 50th iteration. Except for Dirichlet $\alpha = 0.03$, all were considered to reach the optimal play. However, the win rate of Dirichlet $\alpha = 0.03$ (48.44±0.98%) was still near 50%. This showed that Dirichlet α did not influence too much on learning the optimal play.

The AEs for different Dirichlet α's are depicted in Fig. 7. The lowest AE (0.0100±0.0074) was reached at Dirichlet $\alpha =$ 12. The settings of 1.5, 3, and 6 also obtained relatively low AEs. Similar to the observations from c_{puct}, higher levels of exploration (lower Dirichlet α) led to higher AEs.

Fig. 8 shows the total numbers of distinct seen positions during training for different Dirichlet α's. As expected, the number of seen positions decreased as Dirichlet α increased, though the influence was not as obvious as c_{puct}.

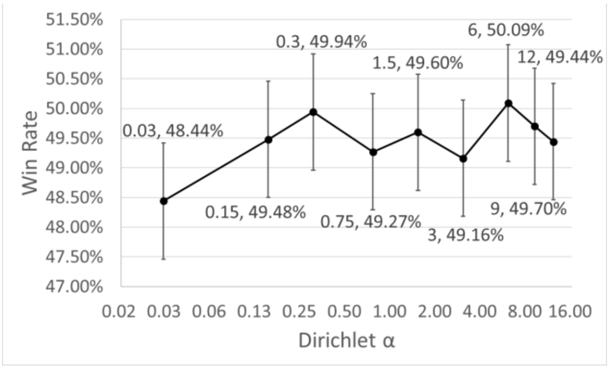

Fig. 6. The win rates at the 50th iteration for different Dirichlet α.

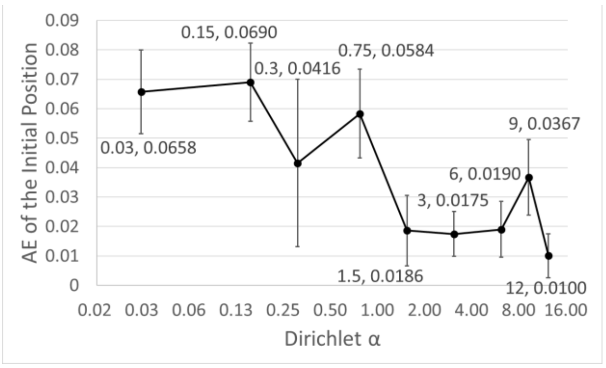

Fig. 7. The averaged AEs of the initial position over the last five iterations for different Dirichlet α.

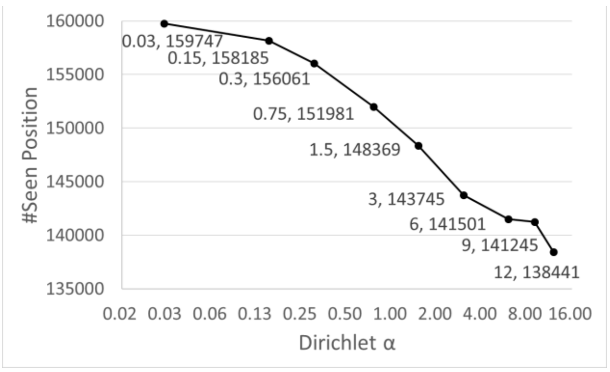

Fig. 8. The total numbers of distinct seen positions during training for different Dirichlet α.

D. Results on Dirichlet ε

The tested values of Dirichlet ε included 0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, and 0.5, where 0.25 was used in AlphaGo Zero and the AlphaZero algorithm [1, 2]. The training curves are also omitted since they did not show much difference. Fig. 9 depicts the win rates achieved by different Dirichlet ε's at the 50th iteration. The range between 0.2 to 0.4 all reached the optimal play. The win rate was the lowest (46.93±0.98%) when the Dirichlet noises were ignored (Dirichlet $\varepsilon = 0$). This demonstrated that the Dirichlet noises did help the learning of the algorithm.

The AEs for different Dirichlet ε's are shown in Fig. 10. Similar to previous observations, higher levels of exploration (higher Dirichlet ε) led to higher AEs. As for the total numbers of distinct seen positions during training, depicted in Fig. 11, the results were as expected. Namely, the number of seen positions increased as Dirichlet ε increased.

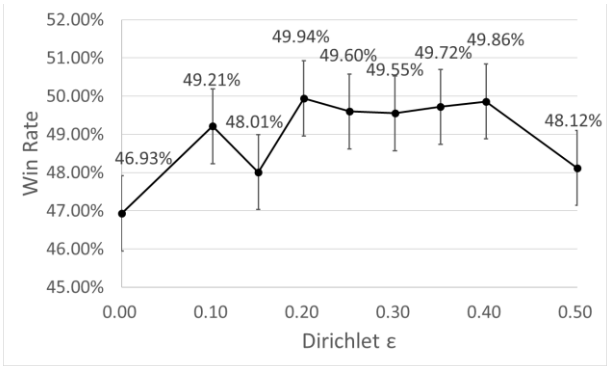

Fig. 9. The win rates at the 50th iteration for different Dirichlet ε.

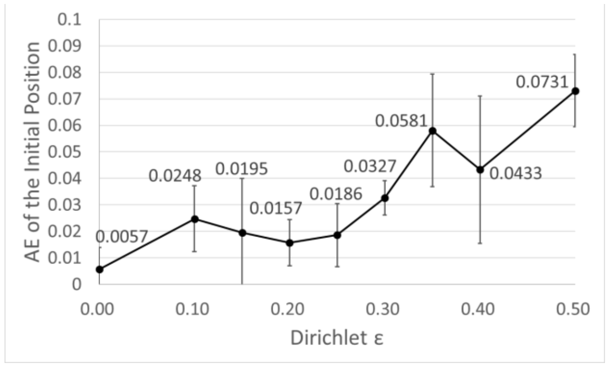

Fig. 10. The averaged AEs of the initial position over the last five iterations for different Dirichlet ε.

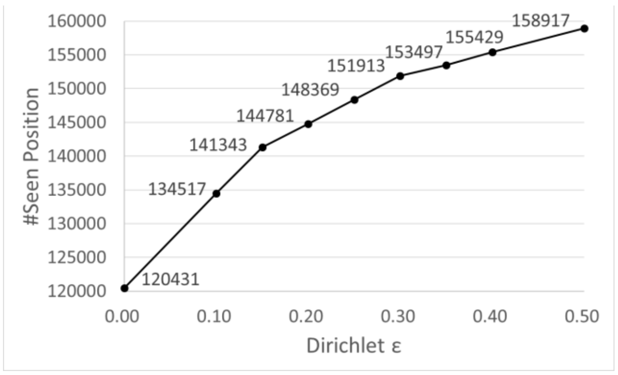

Fig. 11. The total numbers of distinct seen positions during training for different Dirichlet ε.

E. Results on Temperature τ

The tested values for temperature τ included 0.0625, 0.125, 0.25, 0.5, 1, 2, 4, 8, and 16. The results showed that for temperature $\tau \geq 1$, the speed of learning decreased as temperature τ increased (higher levels of exploration). The training curves are not presented due to page limit. The win rates achieved by different temperature τ's at the 50th iteration are shown in Fig. 12. All of the win rates were close to 50%, while only temperature $\tau = 1$ (49.60±0.98%) was considered to reach the optimal play.

Fig. 13 depicts the AEs for different temperature τ's. Similarly, with higher levels of exploration (higher temperature τ), the AEs became higher. The results on the total numbers of distinct seen positions during training, shown in Fig. 14, had similar tendency as c_{puct}. Namely, when the level of exploration was too high, the number of seen positions decreased within the same amount of training games.

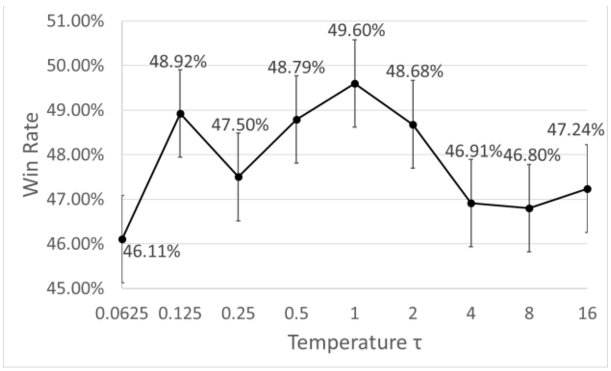

Fig. 12. The win rates at the 50th iteration for different temperature τ.

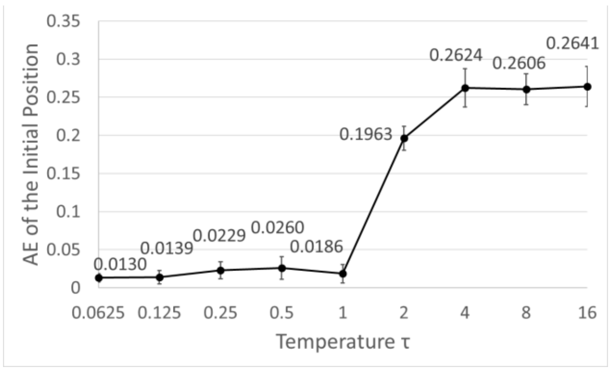

Fig. 13. The averaged AEs of the initial position over the last five iterations for different temperature τ.

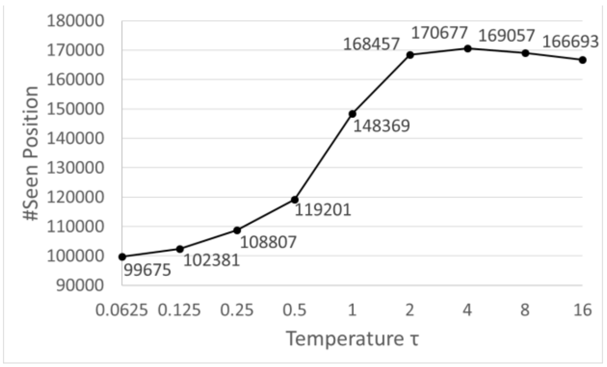

Fig. 14. The total numbers of distinct seen positions during training for different temperature τ.

F. Discussions

In general, the AlphaZero algorithm learned the theoretical value and the optimal play in the material combination "PPPP vs. pppp," unless extreme values of hyper-parameters were used. From Fig. 4, Fig. 7, Fig. 10, and Fig. 13, many of the low AEs indicated that the values derived from the algorithm approximated to the theoretical value. From Fig. 3, Fig. 6, Fig. 9, and Fig. 12, many of these win rates were near 50% against the optimal play. The results demonstrated the robustness of the algorithm.

The four tested hyper-parameters were all related to exploration. Higher c_{puct}'s, Dirichlet ε's, and temperature τ's, or lower Dirichlet α's led to higher levels of exploration. Comparing the total numbers of distinct seen positions during training (Fig. 5, Fig. 8, Fig. 11, and Fig. 14), c_{puct} and temperature τ had more influence on the algorithm than Dirichlet α and Dirichlet ε. However, a consistent tendency was observed in these four hyper-parameters. The AEs tended to increase when the levels of exploration increased beyond some threshold (Fig. 4, Fig. 7, Fig. 10, and Fig. 13).

In addition, for c_{puct} and temperature τ, when the levels of exploration were too high, the number of seen positions decreased (Fig. 5 and Fig. 14). In these two hyper-parameters, it was also observed that the learning speeds decreased as the levels of exploration increased beyond some threshold ($c_{puct} \geq 1$ and temperature $\tau \geq 1$).

The result in Fig. 9 (Dirichlet $\varepsilon = 0$) showed that the AlphaZero algorithm was benefited from Dirichlet noises, though the effect was small. Possible explanations were that the influence of the Dirichlet noises was bounded by c_{puct}, and that the average number of legal moves in 2×4 CDC was too small (~4).

V. Conclusions

This paper applies the AlphaZero algorithm to a solved non-deterministic game 2×4 CDC. Four hyper-parameters related to exploration in the algorithm are experimented. The results demonstrate that the AlphaZero algorithm learns the theoretical value and the optimal play in the material combination "PPPP vs. pppp." To our knowledge, this paper is the first to apply the AlphaZero algorithm to non-deterministic games.

For future work, promising directions include the followings. The first is to conduct experiments on more complicated material combinations in 2×4 CDC to verify whether the AlphaZero algorithm is also able to learn theoretical values and optimal plays well. In addition, experiments can be done to analyze whether the good settings of the hyper-parameters in this paper is applicable to other material combinations. The second is to replace lookup tables by feature extractors such as neural networks and study whether the algorithm produces consistent results. This may provide insights on whether the algorithm can be incorporated to solve games. The third is to investigate the influence of the hyper-parameters to the algorithm more thoroughly, and then compare the values selected by Bayesian optimization.

Acknowledgment

This research is financially supported by the Ministry of Science and Technology of the Republic of China (Taiwan) under contract numbers MOST 106-2221-E-009-139-MY2, 106-2221-E-305-016-MY2, 106-2221-E-305-017-MY2, 107-2634-F-009-011, and 107-2634-F-259-001.

References

[1] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature,* vol. 550, no. 7676, pp. 354-359, 2017, doi: 10.1038/nature24270.

[2] D. Silver *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR,* vol. abs/1712.01815, 2017,

[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," *Proceedings of the IEEE,* vol. 104, no. 1, pp. 148-175, 2016, doi: 10.1109/JPROC.2015.2494218.

[4] B.-N. Chen, B.-J. Shen, and T.-s. Hsu, "Chinese dark chess," *ICGA Journal,* vol. 33, no. 2, pp. 93-106, 2010, doi: 10.3233/ICG-2010-33204.

[5] 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," *Theoretical Computer Science,* vol. 644, pp. 63-75, 2016, doi: 10.1016/j.tcs.2016.06.025.

[6] 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 Journal,* vol. 40, no. 2, pp. 77-104, 2018, doi: 10.3233/ICG-180046.

[7] H.-J. Chang, J.-C. Chen, C.-W. Hsueh, and T.-s. Hsu, "Analysis and efficient solutions for 2×4 Chinese dark chess," *ICGA Journal,* vol. 40, no. 2, pp. 61-76, 2018, doi: 10.3233/ICG-180049.

[8] H.-J. Chang and T.-s. Hsu, "A quantitative study of 2 × 4 Chinese dark chess," in *Computers and Games: 8th International Conference, CG 2013, Yokohama, Japan, August 13-15, 2013, Revised Selected Papers*, H. J. van den Herik, H. Iida, and A. Plaat, Eds. Cham: Springer International Publishing, 2014, pp. 151-162, doi: 10.1007/978-3-319-09165-5_13.

[9] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature,* vol. 529, no. 7587, pp. 484-489, 2016, doi: 10.1038/nature16961.

[10] *gcp/leela-zero: Go engine with no human-provided knowledge, modeled after the AlphaGo Zero paper*. Available: https://github.com/gcp/leela-zero

[11] *tensorflow/minigo: An open-source implementation of the AlphaGoZero algorithm*. Available: https://github.com/tensorflow/minigo

[12] *ELF | Game Research Platform | Facebook AI*. Available: https://facebook.ai/developers/tools/elf

[13] C.-H. Hsueh and I.-C. Wu, "DarkKnight wins Chinese dark chess tournament," *ICGA Journal,* vol. 38, no. 4, pp. 249-251, 2015, doi: 10.3233/ICG-2015-38411.

[14] W.-J. Tseng, J.-C. Chen, and I.-C. Wu, "DarkKnight wins Chinese dark chess tournament," *ICGA Journal,* vol. 39, no. 2, pp. 163-165, 2017, doi: 10.3233/ICG-170023.

[15] H. J. van den Herik, A. Plaat, and J. Hellemons, "The 16th Computer Olympiad," *ICGA Journal,* vol. 35, no. 1, pp. 50-50, 2012, doi: 10.3233/ICG-2012-35111.

[16] S.-J. Yen, J.-C. Chen, B.-N. Chen, and W.-J. Tseng, "DarkKnight wins Chinese dark chess tournament," *ICGA Journal,* vol. 36, no. 3, pp. 175-176, 2013, doi: 10.3233/ICG-2013-36315.

[17] S.-J. Yen, S.-Y. Chiu, and I.-C. Wu, "MoDark wins the Chinese dark chess tournament," *ICGA Journal,* vol. 33, no. 4, pp. 230-231, 2010, doi: 10.3233/ICG-2010-33410.

[18] *Results | ICGA*. Available: https://icga.org/?page_id=2050#chindarkchess

[19] *ICGA Computer Olympiad 2018*. Available: https://www.tcga.tw/icga-computer-olympiad-2018/en/

[20] J.-C. Chen, T.-Y. Lin, B.-N. Chen, and T.-s. Hsu, "Equivalence classes in Chinese dark chess endgames," *IEEE Transactions on Computational Intelligence and AI in Games,* vol. 7, no. 2, pp. 109-122, 2015, doi: 10.1109/TCIAIG.2014.2317832.

[21] C. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[22] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *From Natural to Artificial Neural Computation*, Berlin, Heidelberg, 1995, pp. 195-201: Springer Berlin Heidelberg, doi: 10.1007/3-540-59497-3_175.