Exact-Win Strategy for Overcoming AlphaZero

Yen-Chi Chen[1]
National Taiwan Normal University
No.88, Sec. 4, Tingzhou Rd., Taipei, Taiwan, R.O.C.
+886 (2) 7734-6665
zxkyjimmy@gmail.com

Chih-Hung Chen[1]
National Taiwan Normal University
No.88, Sec. 4, Tingzhou Rd., Taipei, Taiwan, R.O.C.
+886 (2) 7734-6665
honhonzone@gmail.com

Shun-Shii Lin[1*]
National Taiwan Normal University
No.88, Sec. 4, Tingzhou Rd., Taipei, Taiwan, R.O.C.
+886 (2) 7734-6671
linss@csie.ntnu.edu.tw

ABSTRACT
The Monte-Carlo Tree Search used in the AlphaZero may easily miss a critical move because it is based on sampling search space and focuses on the most promising moves. In addition, the Monte-Carlo Tree Search may sample a move for many times even if this move has been explored with a determined game-theoretical value. In this paper, we propose an Exact-win-MCTS that makes use of sub-tree's information (WIN, LOSS, DRAW, and UNKNOWN) to prune unneeded moves to increase the opportunities of discovering the critical moves. Our method improves and generalizes some previous MCTS variations as well as the AlphaZero approach. The experiments show that our Exact-win-MCTS substantially promotes the strengths of Tic-Tac-Toe, Connect4, and Go programs especially. Finally, our Exact-win Zero defeats the Leela Zero, which is a replication of AlphaZero and is currently one of the best open-source Go programs, with a significant 61% win rate. Therefore, we are pleased to announce that our Exact-win-MCTS has overcome the AlphaZero approach without using extra training time, playing time, or computer resources. As far as we know, this is the first practical idea with concrete experiments to beat the AlphaZero approach.

CCS Concepts
• **Theory of computation→Game tree search**

Keywords
Exact-win-MCTS; MCTS pruning; AlphaZero; Exact-win Zero; Leela Zero.

1. INTRODUCTION
The Monte-Carlo Tree Search (MCTS), a best-first search, is different from the adversarial search which is also known as the miniMax search that considers all possible moves from a given state. The MCTS makes good use of sampling search space to converge to the optimal solution and can handle large search spaces with high branching factors. Since the MCTS not only exploits the most promising sub-trees but also explores the potential good moves, machine learning and the MCTS combined with extensive training is suitable for complex applications and has been a great success in computer Go.

Go had been regarded as the most complicated game in artificial intelligence for a long time. In March 2016, the AlphaGo has defeated the world's top Go player Lee Se-dol that was a major milestone in artificial intelligence research. The AlphaGo's victory has encouraged the AI community that many researchers imitate the AlphaGo [1] (AlphaGo Zero [2] or AlphaZero [3]) algorithm. Professor Shun-Shii Lin[1][2] was instead trying to improve the performance of the AlphaZero when some researchers were reproducing the AlphaZero approach. In early 2018, Professor Lin proposed the B.B.Q.-MCTS (Big, Best, and Quick-win strategies in Monte-Carlo Tree Search) [4][5] idea to try to solve the three key issues of the AlphaZero approach. Firstly, the AlphaZero only judges win, draw, or loss but ignores how many points it will win or lose in a game. Secondly, the Monte-Carlo Tree Search, used in the AlphaZero, derives an average value for a node as the representative of all its children nodes. Thirdly, the AlphaZero never estimates the depth rewards during the Monte-Carlo Tree Search procedure.

Because the MCTS is based on sampling search space and focuses on the most promising moves, it would easily miss a critical move that has not been expanded yet. In addition, a move will be repeatedly visited with high probability if it has been proven as the game-theoretical value. In this paper, we propose another improvement, called Exact-win-MCTS, that can prune unneeded moves for exploring the game tree (like the α-β heuristic within the miniMax algorithm [6]). It means that the Exact-win-MCTS makes use of sub-tree's information to eliminate those branches with a determined game-theoretical value. After months of work, we are delighted to announce the promising results that we have significantly improved the strengths of the Tic-Tac-Toe, Connect4, and even Go with the strongest weights trained by Leela Zero[3].

This paper is organized as follows. Section 2 introduces the background of the MCTS algorithm. Section 3 gives a brief overview of related works on the MCTS. Section 4 describes the Exact-win-MCTS to prune unneeded moves in the MCTS and AlphaZero, and Section 5 shows the results of our experiments. Conclusions and future works follow in Section 6.

CIIS 2018, November 17-19, 2018, Phuket, Thailand

ACM ISBN 978-1-4503-6595-6/18/11…$15.00
DOI: https://doi.org/10.1145/3293475.3293486

[1] These authors contributed equally to this work.

[2] Professor Shun-Shii Lin, department of Computer Science and Information Engineering, National Taiwan Normal University.

* Corresponding author: Shun-Shii Lin.

[3] Leela Zero is available from https://github.com/gcp/leela-zero and the weights trained by Leela Zero is available from https://zero.sjeng.org/networks/b0841a68b4beae9314e47757b44dbe16c5d421dfadcba9ddfe25373e8c27b262.gz. [Accessed 4 August 2018]

2. BACKGROUND

The Monte-Carlo Tree Search is a best-first search based on sampling search space to converge to the miniMax tree. It means that UCT (Upper Confidence Bound 1 applied to trees) leads the MCTS to approximate the optimal solution if enough time and memory are given [7][8]. The MCTS does not need an evaluation function, it estimates a state by averaging the results of simulations after that state or the estimated results from a pre-trained neural network. This algorithm works by repeating the following four steps to build a search tree until the time, memory or iteration constraint is reached. Then the move with the highest reward or the most visits is chosen as the output answer [9]. The four steps of the MCTS are as follows.

A. Selection: start from the root node *R* to descend through the tree until a leaf node *L* is reached.

B. Expansion: expand one (or more) child node C_i from *L*.

C. Simulation: play a random playout from C_i until the end of the game.

D. Backpropagation: update the final game result from C_i to the root node *R*.

3. RELATED WORKS

In this section, we introduce the related works on recording and updating the game-theoretic values in the Monte-Carlo Tree Search (MCTS). The following subsections give a brief introduction about the MCTS-Solver and the MCTS-miniMax hybrids.

3.1 MCTS-Solver

The MCTS-Solver [10] allows the MCTS to backpropagate not only regular simulation outcomes, i.e. -1 for a loss, 0 for a draw, and 1 for a win, but also game-theoretic values, i.e. $-\infty$ for a proven LOSS and ∞ for a proven WIN whenever terminal states are encountered during the game tree search. However, their method cannot deal with the proven DRAW outcomes. A state will be marked as a proven LOSS ($-\infty$) if one of its children has been marked as a proven WIN (∞). On the other hand, a state will be marked as a proven WIN (∞) if all of its children have been marked as proven LOSS ($-\infty$). The MCTS-Solver is good at handling game-theoretic values than the traditional MCTS because it avoids wasting time on the re-sampling of some proven game states. But the MCTS-Solver still suffers from the following situation. As mentioned in [10], in the case that only some children have been marked as proven LOSS ($-\infty$), but one or more children of the node have a different value, we cannot mark the node as a proven WIN (∞). Therefore, the MCTS-Solver will propagate -1, the simulation result for a loss game, instead of $-\infty$, the game-theoretical value for those LOSS children. Then in the later selection phases, those LOSS children nodes may be re-sampled many times before all the other children have been proven to be LOSS. In this situation, our Exact-win-MCTS will not select those LOSS children anymore in the later selection phases. Instead, our approach will select other children for expansion.

Furthermore, their method is not easy to be generalized to the kind of games with draw outcomes that need a specified value to denote the proven DRAW. Their method also does not fit for AlphaZero improvement.

3.2 MCTS-MiniMax Hybrids

The MCTS-miniMax Hybrids [11][12] was proposed by Baier and Winands that combined the miniMax and the MCTS in three different phases: the rollout phase, called MCTS-MR for MCTS with miniMax Rollouts; the selection and expansion phases, called MCTS-MS for MCTS with miniMax Selection; and the backpropagation phase, called MCTS-MB for MCTS with miniMax Backpropagation. We give a brief introduction about the MCTS-miniMax hybrids as follows.

A. MiniMax in the Rollout phase

Since the traditional MCTS's convergence to the optimal policy is very slow, smarter rollout strategies would be effective in improving its performance [13]. For that reason, a shallow miniMax search is used for enhancing rollout phase. Because of without evaluation function, the miniMax search used in the MCTS-MR can just find a forced WIN and a forced LOSS. For example, a miniMax search is started for the first rollout move *a*. If move *a* loses this game, then the next move *b* will be chosen to start another miniMax search. Finally, a random move will be returned if the miniMax search cannot find any WIN or LOSS path. The MCTS-MR strategy improves the quality of rollout phase by avoiding shallow traps.

B. MiniMax in the Selection and Expansion phases

The miniMax search can be triggered everywhere during the tree traversal in the selection and expansion phases of the MCTS. A miniMax search is started when a state has reached a given number of visits. The MCTS-MS checks nodes by a shallow and wide miniMax search to guide the MCTS tree growth by detecting shallow wins and avoiding shallow losses. If the miniMax search has proven a WIN or LOSS then this value can be updated up to the root. Otherwise, the selection and expansion phase continue as the traditional MCTS.

C. MiniMax in the Backpropagation phase

As mentioned in Section 3.1, the MCTS-Solver tries to update the game-theoretic values to ascend through the tree. A state is proven to be a LOSS if one of its children is a proven WIN. On the other hand, a state is proven to be a WIN if all of its children are proven to be LOSS. When encountering an unproven state, the MCTS-Solver will be stopped and will be switched to the traditional MCTS. Therefore, the MCTS-MB triggers a shallow miniMax search to try to prove all the moves after the given state. The MCTS-MB allows the MCTS to prove a WIN/LOSS more quickly and exclude useless moves effectively.

4. EXACT-WIN-MCTS

The extraordinary success of the AlphaGo let many AI researchers re-implement the AlphaGo [1], AlphaGo Zero [2], or AlphaZero [3]. However, our group tries to find out the inadequacies of the AlphaGo Zero (AlphaZero) and proposes the B.B.Q.-MCTS (Big, Best, and Quick-win strategies in Monte-Carlo Tree Search) [4][5] to try to solve them. As a result, the preliminary experiments show that our B.B.Q.-MCTS works pretty well for improving the performance of the AlphaZero approach for sudden death games and 6x6 Othello. Even so, there are still some other problems within the MCTS used in the AlphaZero. It would easily miss a critical move because of focusing on the most promising moves. Another problem is that a move will be re-sampled frequently even if the move has been proven to be a WIN or LOSS. In this paper, we propose the Exact-win-MCTS to cut those moves with exactly WIN or LOSS results.

It can avoid re-sampling those proven moves to increase an opportunity of exploring the critical move. The detail of our Exact-win-MCTS is described as follows.

Our Exact-win-MCTS makes use of previous information to prune unneeded moves for exploring the unknown space of a game tree. So we need to know which moves exactly win or lose the game, i.e. a move has a certain game-theoretical value if it has been proven to be a WIN or LOSS. To accomplish the Exact-win-MCTS, we add an additional label to mark the state of a node where it has four states: WIN, LOSS, DRAW, and UNKNOWN. The four steps of our Exact-win-MCTS are modified as follows:

A. Selection: start from the root node R to descend through the tree until an unmarked leaf node L_u is reached. In this paper, we modify UCB policy to fit the Exact-win-MCTS. Our UCB policy is modified as:

$$\text{Let } UC(s) = \{a \in Action(s) | (s,a) \text{ is unmarked}\} \quad (1)$$

$$UCB = argmax_{a \in UC(s)} \left(\frac{W(s,a)}{N(s,a)} + c\sqrt{\frac{\sum_p N(s,p)}{N(s,a)}} \right) \quad (2)$$

B. Expansion: expand one (or more) child node C_i from L_u. In this work, we only expand one child node for each simulated game.

C. Simulation: play a random playout from C_i until the end of the game.

D. Backpropagation: update the final game results from C_i to the root node R according to Algorithm 1.

Algorithm 1. Backpropagation of the Exact-win-MCTS

```
1.  If has-win-child
2.      mark LOSS
3.  If unknown-children-count == 0
4.      If has-draw-child
5.          mark DRAW
6.      Else
7.          mark WIN
```

The MCTS-Solver [10] uses integers to represent not only regular simulation results such as loss (-1), draw (0) and win (+1), but also game-theoretic values such as proven LOSS ($-\infty$) and proven WIN ($+\infty$) in the search tree. In addition to marking a state with WIN, LOSS, and regular simulation results, our Exact-win-MCTS adds additional labels to mark a state with DRAW and UNKNOWN, which are not easily be tackled with some corresponding integer values such as 0 or something else in the MCTS-Solver. Thus, our Exact-win-MCTS makes the game information more complete and let our approach can correctly deal with the games with DRAW outcomes.

Because the MCTS-Solver may overestimate or underestimate the value of a node as mentioned in the paper [10], Baier and Winands proposed the MCTS-Solver plus MCTS-MB [11][12] to ease these estimation problems. But some nodes that have been proven may still be revisited lots of times by a miniMax search in the MCTS-MB. Moreover, the information of simulations (win rate) may be changed by the game-theoretical values in the MCTS-Solver plus MCTS-MB. Figure 1 is an example to explain the difference between our Exact-win-MCTS and the MCTS-Solver plus MCTS-MB. The node D in Figure 1 has been proven to be a WIN, so its parent node B is proven to be a LOSS. The node E can be pruned by the MCTS-Solver, but MCTS-MB will do a shallow miniMax search to revisit nodes B, D, and E. In the next round of simulation, the MCTS-Solver will re-sample those nodes (C, F, and G) that have been visited by the previous miniMax search. In this case, our Exact-win-MCTS will cut node E and choose node C in the next round of simulation. It means that Exact-win-MCTS doesn't re-sample (or revisit) nodes that have been proven.

The nodes that have been proven will be pruned in our Exact-win-MCTS, but may be revisited lots of times by a miniMax search in the MCTS-MB. So our Exact-win-MCTS can explore more unknown space of a game tree than the previous methods. Moreover, a miniMax search in the MCTS-MB may make vain efforts at the opening, even the middle stage of the game, because the situation of a game is still unclear.

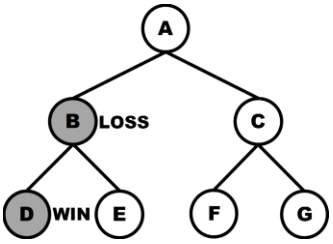

Figure 1. A schematic diagram for comparing between our Exact-win-MCTS and the MCTS-Solver plus MCTS-MB.

Our Exact-win-MCTS is a pruning algorithm or endgame solver based on the traditional MCTS. It is a more general, complete and efficient algorithm which achieves the capacity as the MCTS-Solver plus the MCTS-MB, but uses fewer computation resources and keeps the simulation results of the MCTS unchanged. The unchanged simulation results mean that the win rate of our Exact-win-MCTS doesn't update by $\pm\infty$, so we can keep the original information of simulations. Furthermore, it can tackle the kind of games with DRAW outcomes. Table 1 lists the comparison between our Exact-win-MCTS and the MCTS-Solver plus MCTS-MB.

Table 1. Comparison between our Exact-win-MCTS and the MCTS-Solver plus MCTS-MB

	Exact-win-MCTS	**MCTS-Solver + MCTS-MB**
Node states	WIN, LOSS, DRAW, and UNKNOWN	WIN and LOSS
Nodes have been proven	Prune	May revisit
Search in vain	No	Maybe
Draw games	Can tackle	Cannot tackle
Simulation results	Remain unchanged	May be changed

5. EXPERIMENTS

In this section, we apply the Exact-win-MCTS to Tic-Tac-Toe, Connect4, and even 19×19 Go. We compare the Exact-win-MCTS against the traditional MCTS. The experimental results are shown in the following subsections.

5.1 Tic-Tac-Toe

We pick two Tic-Tac-Toe game situations, as shown in Figure 2, where each will have an obvious result. In Figure 2 (a), it is the player "O"'s turn to play and he/she will be losing the game no matter where he/she plays. Table 2 shows that our Exact-win-

MCTS can exactly estimate the situation by repeating just 53 simulations. In Figure 2 (b), it is the player "X"'s turn to play and he/she will be winning the game unless he/she plays at B2 so that a draw is gotten. Table 3 shows that our Exact-win-MCTS can exactly estimate the situation by repeating just 113 simulations. Those results show that our Exact-win-MCTS uses much smaller numbers of simulations to reach an exact outcome easily, but the traditional MCTS can't. Note that our Exact-win-MCTS for Tic-Tac-Toe here needs no human knowledge, evaluation function, or pre-trained neural network model.

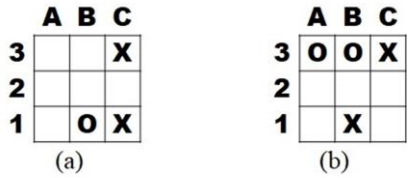

Figure 2. Two Tic-Tac-Toe situations with obvious results for testing the traditional MCTS and Exact-win-MCTS.

Table 2. Comparison between our Exact-win-MCTS and the traditional MCTS on Figure 2 (a)

Position	A1	A2	A3	B2	B3	C2
MCTS (win rate/ simulations)	7.5%/ 20	17.1%/ 35	9.1%/ 22	19.0%/ 42	15.6%/ 32	21.4%/ 49
Exact-win-MCTS (win rate/ simulations)	0%/ 6	0%/ 5	0%/ 5	0%/ 5	0%/ 5	0%/ 27

Table 3. Comparison between our Exact-win-MCTS and the traditional MCTS on Figure 2 (b)

Position	A1	A2	B2	C1	C2
MCTS (win rate/ simulations)	87.5%/ 56	79.3%/ 29	70.6%/ 17	80.0%/ 35	89.7%/ 63
Exact-win-MCTS (win rate/ simulations)	100%/ 13	100%/ 41	50%/ 21	100%/ 15	100%/ 23

Finally, we let our Exact-win-MCTS do self-play from an initially empty board of Tic-Tac-Toe. The results show that playing the 1st move at one in the middle of outer positions (A2, B1, B3, and C2) is the best choice. That makes sense because the sub-trees of those moves are more complicated than playing at other positions, such as B2.

5.2 Connect4

We choose a Connect4 game situation, as shown in Figure 3, which has an obvious result. It is the player "O"'s turn to play and he/she will be losing the game no matter where he/she plays. In this work, our Exact-win-MCTS is implemented also without any human knowledge, evaluation function, or pre-trained neural network model. Table 4 shows that our Exact-win-MCTS can exactly estimate the situation by repeating just 16576 simulations, but the traditional MCTS can't even with 20000 simulations.

5.3 Go

Go is a very complex game with 2×10^{170} legal board positions estimated by Tromp and Farnebäck [14]. It is very difficult to develop a high strength Go program all by ourselves. As a consequence, we prefer to seek an open source and then choose the Leela Zero which is one of the most famous Go projects. Leela Zero, a fairly faithful reimplementation of the AlphaGo Zero [2], desires to recompute the AlphaGo Zero weights which will take about 1700 years on a commodity machine. Smartly, they called for lots of volunteers' computers to continuously train the weights of their neural network. Because of the limitation of the resource and time, we also make use of the weights[4] already trained by Leela Zero.

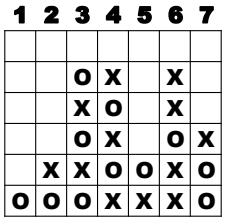

Figure 3. A Connect4 game with obvious results for testing the traditional MCTS and Exact-win-MCTS.

Table 4. Comparison between our Exact-win-MCTS and the traditional MCTS on Figure 3

Position	1	2	3	4	5	6	7
MCTS (win rate/ simulations)	24.9%/ 1928	28.3%/ 16796	9.8%/ 123	19.5%/ 452	12.8%/ 168	15.6%/ 243	16.9%/ 290
Exact-win-MCTS (win rate/ simulations)	0%/ 5299	0%/ 4454	0%/ 913	0%/ 4019	0%/ 6	0%/ 1354	0%/ 531

In this work, we apply the Exact-win-MCTS to the Leela Zero, named Exact-win Zero, to compare with the original Leela Zero. We manually design an interesting endgame of Go (see Figure 4) to observe the behavior of the Leela Zero and Exact-win Zero respectively. Figure 5 shows that the Leela Zero moves gradually towards the right win rates' estimation for both Black and White sides starting from the 27th move. That means the Leela Zero sees the result of this endgame after the 27th move. However, our Exact-win Zero sees the result of this endgame after the 11th move, as shown in Figure 6. It means that the horizon effect [15] of the Leela Zero is coming much earlier than our Exact-win Zero. In other words, our Exact-win Zero can search much deeper than the Leela Zero.

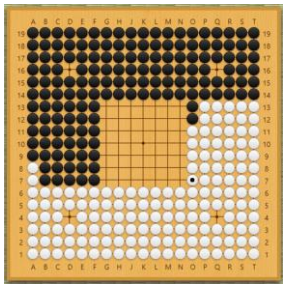

Figure 4. A manually made endgame of Go.

[4] We use the newest weights of Leela Zero on 2018-08-04. The version of this weights is with a hash code b0841a68b4beae9314e47757b44dbe16c5d421dfadcba9ddfe25373e8c27b262.

Finally, we let our Exact-win Zero play 100 games against the Leela Zero starting from an initially empty board, each being the Black player for 50 games. In general, two programs with the same strength (weights) playing to each other would get a fifty-fifty result. But our Exact-win Zero wins 61 games among 100 games against the Leela Zero. Hence, the win rate of our Exact-win Zero is 61% as shown in Table 5. The result is trustworthy with the standard deviation greater than 2 times of 100 games. It ensures that our Exact-win-MCTS is very effective in improving the strength of AlphaZero approach.

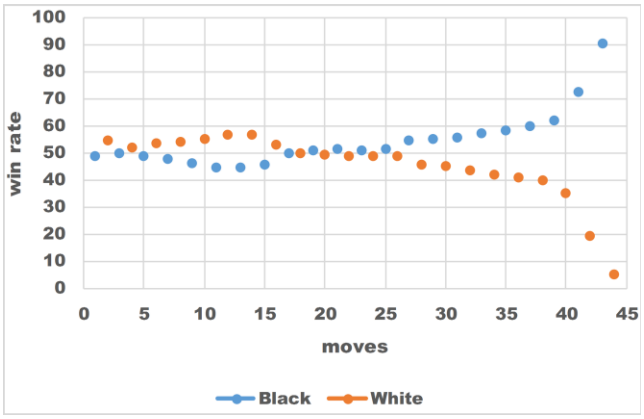

Figure 5. Estimation of Figure 4 by the Leela Zero.

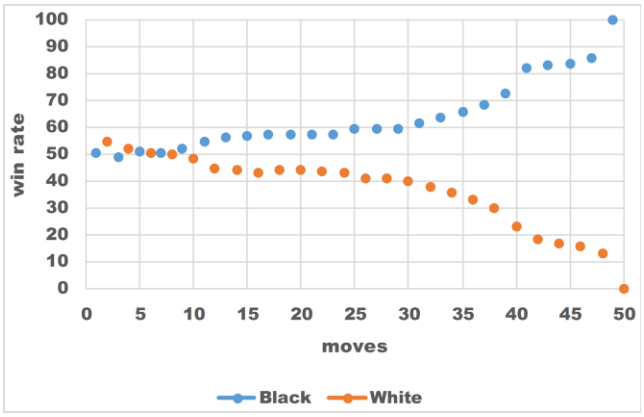

Figure 6. Estimation of Figure 4 by the Exact-win Zero.

Table 5. Our Exact-win Zero plays against the Leela Zero

	Exact-win Zero	Leela Zero	Win rate
Exact-win Zero	-	61	61%
Leela Zero	39	-	39%

In the experiment of Table 5, no matter who (the Exact-win Zero or the Leela Zero) plays Black or White, the win rates of Black and White are 49% and 51% respectively. This makes sense because White has a little more advantage on Go because of the 7.5 komi. Among the 100 games, our Exact-win Zero playing Black (resp. White) gets a 60% (resp. 62%) win rate, as shown in Figure 7. Indeed, our Exact-win Zero always outperforms the Leela Zero no matter it plays Black or White.

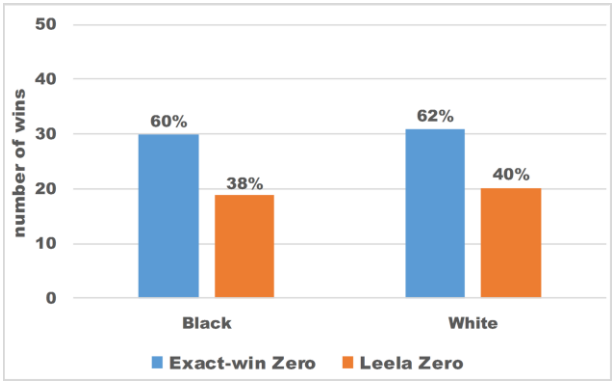

Figure 7. The win rate of Black and White among 100 games.

Among these 100 games, every game was over with at least 151 moves and with at most 416 moves. According to how many moves were played for each game, we classify these games into 4 sets: less than 201 moves, 201~250 moves, 251~300 moves, and more than 300 moves. There are 18, 31, 24 and 27 games in the set of games with less than 201 moves, 201~250 moves, 251~300 moves, and more than 300 moves respectively. Our Exact-win Zero wins 11, 20, 15 and 15 games in these 4 sets respectively. The win rates of our Exact-win Zero vs. the Leela Zero for each set are shown in Figure 8. The results show that our Exact-win Zero plays stronger in each set with a win rate of at least 55.6%.

From the algorithmic point of view, our approach takes advantage of avoiding re-sampling those proven nodes with exactly win or loss results and can explore more critical nodes in the game tree. Hence, the Exact-win Zero can see clearer than the Leela Zero at the uncertain opening and middle stages of the games. At the endgame stage, the game results are more certain and both the Exact-win Zero and the Leela Zero are less likely to make the wrong moves. But even so, our Exact-win Zero can still see clearer than the Leela Zero since our Exact-win Zero gets a 55.6% win rate in the set of games with more than 300 moves.

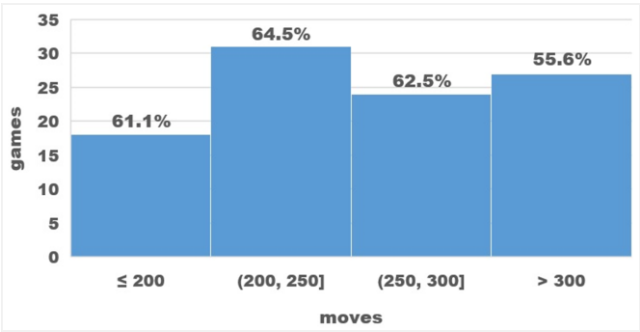

Figure 8. The win rates of Exact-win Zero for each set among 100 games.

6. CONCLUSIONS AND FUTURE WORKS

The previous experiments show that our Exact-win-MCTS significantly improves the performance of sudden death games, such as Tic-Tac-Toe and Connect4, without training a neural network. In addition, our Exact-win-MCTS also significantly improves the strengths on non-sudden death games, such as Go, which has a pre-trained neural network. The promising results ensure that our Exact-win Zero defeats the Leela Zero with a

significant 61% win rate. In conclusion, our Exact-win-MCTS has already overcome the AlphaZero approach.

It is also possible to apply our method for the training stage of AlphaZero. However, this needs lots of computer resources, such as GPUs, TPUs, computers, and lots of time to train the neural network to achieve a top-level weight like AlphaGo Zero's weight. This deserves to be tried and tested in the future. By using our Exact-win-MCTS for the training stage, we expect that the final weights of the neural network will be much better than previous weights made by those top-leveled organizations such as Deepmind, Leela, Facebook, or Tencent.

7. ACKNOWLEDGMENTS

This research was supported by the Ministry of Science and Technology (R.O.C.) under grants MOST 106-2221-E-003-027-MY2 and MOST 107-2634-F-259-001.

8. REFERENCES

[1] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 7587 (2016), 484-489. DOI= https://doi.org/10.1038/nature16961.

[2] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T. and Hassabis, D. 2017. Mastering the game of Go without human knowledge. *Nature*, 550, 7676 (2017), 354-359. DOI= https://doi.org/10.1038/nature24270.

[3] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. and Hassabis, D. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *Arxiv.org* (December 5, 2017). [online] Available at: https://arxiv.org/pdf/1712.01815.pdf [Accessed July 10, 2018].

[4] Chang, N.-Y., Chen, C.-H., Lin, S.-S. and Nair, S. 2018. The Big Win Strategy on Multi-Value Network: An Improvement over AlphaZero Approach for 6x6 Othello. In *Proceedings of the 2018 International Conference on Machine Learning and Machine Intelligence* (Hanoi, Vietnam, September 28-30, 2018). DOI= https://doi.org/10.1145/3278312.3278325.

[5] Chen, C.-H., Wu, W.-L., Chen, Y.-H. and Lin, S.-S. 2018. Some Improvements in Monte Carlo Tree Search Algorithms for Sudden Death Games. In *Proceedings of the 10th International Conference on Computers and Games* (Taipei, Taiwan, July 9-11, 2018). DOI= https://doi.org/10.3233/ICG-180065.

[6] Edwards, D.J. and Hart, T.P. 1961. The Alpha-Beta Heuristic. *Dspace.mit.edu* (1961). [online] Available at: http://dspace.mit.edu/handle/1721.1/6098 [Accessed September 17, 2018].

[7] Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4, 1 (2012), 1-43. DOI= https://doi.org/10.1109/TCIAIG.2012.2186810.

[8] Kocsis, L. and Szepesvari, C. 2006. Bandit Based Monte-Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning* (Berlin, Germany, September 18-22, 2006), Lecture Notes in Computer Science. Furnkranz, J., Scheffer, T. and Spiliopoulou, M. (Eds.). *Springer*, 4212 (2006), 282-293. DOI= https://doi.org/10.1007/11871842_29.

[9] Chaslot, G., Winands, M., Herik, H., Uiterwijk, J. and Bouzy, B. 2008. Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, 4, 3 (2008), 343-357. DOI= https://doi.org/10.1142/S1793005708001094.

[10] Winands, M.H.M., Björnsson, Y., Saito, JT. 2008. Monte-Carlo Tree Search Solver. In *Proceedings of the 6th International Conference on Computers and Games*, Lecture Notes in Computer Science. van den Herik, H.J., Xu X., Ma, Z., Winands, M.H.M. (Eds.). *Springer*, 5131 (2008), 25-36. DOI= https://doi.org/10.1007/978-3-540-87608-3_3.

[11] Baier, H. and Winands, M. 2015. MCTS-Minimax Hybrids. *IEEE Transactions on Computational Intelligence and AI in Games*, 7, 2 (2015), 167-179. DOI= https://doi.org/10.1109/tciaig.2014.2366555.

[12] Baier, H. and Winands, M. 2013. Monte-Carlo Tree Search and Minimax Hybrids. In *the IEEE Conference on Computational Intelligence and Games* (Niagara Falls, Canada, August 11-13, 2013). DOI= https://doi.org/10.1109/CIG.2013.6633630.

[13] Gelly, S., Wang, Y., Munos, R. and Teytaud, O. 2006. Modification of UCT with Patterns in Monte-Carlo Go. [Research Report] RR-6062, *Inria* (France, 2006). [online] Available at: https://hal.inria.fr/inria-00117266v3 [Accessed September 17, 2018].

[14] Tromp, J., Farnebäck, G. 2007. Combinatorics of Go. In *Proceedings of the 5th International Conference on Computers and Games*, Lecture Notes in Computer Science. van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M. (Eds.). *Springer*, 4630 (2006), 84-99. DOI= https://doi.org/10.1007/978-3-540-75538-8_8.

[15] Berliner, H. 1973. Some Necessary Conditions for a Master Chess Program. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence* (Stanford, CA, USA, August 20 –23, 1973), 77-85.