Single Conspiracy Number Analysis in Checkers

Sarot Busala[1], Zhang Song[1], Hiroyuki Iida[1], Mohd Nor Akmal Khalid[1]
and Umi Kalsom Yusof[2]

[1]*Japan Advanced Institute of Science and Technology, 923-1211 Ishikawa Prefecture, Nomi, Asahidai, Japan*

[2]*School of Computer Sciences, University of Science Malaysia, 11800 Georgetown, Pulau Pinang, Malaysia*

Keywords: Checkers, MIN-MAX Search Tree, Conspiracy Number, Single Conspiracy Number.

Abstract: Game playing provides the medium for a variety of algorithms to formulate play decisions that surpass human expert. However, the reasons that distinguish the winning and losing positions remain actively researched which leads to the utilization of the search "indicator". Conspiracy number search (CNS) and proof number search (PNS) had been popularly adopted as the search indicators in MIN/MAX and AND/OR tree, respectively. However, their limitations had encouraged the need for an alternative search indicator. The single conspiracy number (SCN) is a search indicator inspired by CNS and PNS which measure the difficulty of getting MIN/MAX value over a threshold point for a current root node. Recently, SCN had been successfully applied in Chinese chess to analyze both the progress pattern and long-term position. In this paper, analysis of the SCN within the game of checkers is conducted where different SCN threshold values and varying depth of the search tree were considered. Checkers was chosen due to its smaller search space and contain a rule that affects the shape of the search tree. The experimental results show that the SCN values stabilize as the depth of the search tree increases whether the player is in winning, drawing or losing position.

1 INTRODUCTION

When playing games, human players use analytic skill and intuition that built up from their experience. When expert players play the game, they tend to play faster, better and have more stability than beginner players. Recently, the AlphaZero algorithm achieved better performance by a large margin in the game of chess, shogi and Go without any domain knowledge except the rules of the game (Silver et al., 2017). Although the algorithm succeeded in selecting better play decisions, the reasons or properties that distinguish those plays from best and worse are still actively researched. This paper explores the properties that distinguish an algorithm's performance in the checkers game.

The similarity of chess, shogi, Go and checkers games are the information of their states which are always available for both players (called perfect information games). This information can be analyzed and evaluated to distinguish the winning and losing positions based on the game progress. The mechanics of board games involve using a tree searching algorithm to evaluate and decide the possible moves to take (Khalid et al., 2015b). However, even in the best-known search algorithms, the search space may grow exponentially complex with the growing depth of the tree. In addition, there is no point of continuing the game when the "indication" of the final outcome can be estimated (Al-Khateeb and Kendall, 2012). As such, identifying the properties that distinguish winning and losing positions is highly dependent on the quantifying capabilities of the adopted indicator.

Single Conspiracy Number (SCN) is a variation of a well-known search indicator called conspiracy number search (CNS) (McAllester, 1988) and its successive search indicator called proof number search (PNS) (Allis et al., 1994). SCN tries to evaluate the difficulty of the current state of the game by getting the MIN/MAX value over the specified threshold point. SCN is calculated during the decision process of the game progress where the difficulty of the current state is measured. If the game states can be search completely, SCN will show the possibility of the current state getting a score higher than a specified threshold value in a MIN/MAX search tree. However, most games like Chess or Go have enormous search space for all of its possible states. As such, the search algorithms in the games employ heuristic evaluation functions and pruning techniques to reduce the search

Busala, S., Song, Z., Iida, H., Khalid, M. and Yusof, U.
Single Conspiracy Number Analysis in Checkers.
DOI: 10.5220/0007344005390546
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 539-546
ISBN: 978-989-758-350-6

space to a manageable level. Since the SCN is calculated based on an explored search tree, it also gets affected by the shape of the search tree and the nodes explored.

This paper focuses on SCN in the checkers game which is calculated in multiple situations and their effect in each situation is determined. The experiment was conducted in an open-source checker environment named Samuel. The SCNs of winning, drawing and losing position was discussed separately to observe the differences property of SCN in each situation. The remainder of the paper is organized as follows. First, Section 2 reviews of the previous works and Section 3 provide details of the method. In Section 4 the results of an experiment conducted with two competing homogenous Samuels are collected and analyzed. Lastly, Section 5 concludes this study.

2 LITERATURE REVIEW

The MIN/MAX search algorithm has been applied to various problems which include decision making and game theory for two-player perfect information games such as Tic-Tac-Toe, Chess, etc. The MIN/MAX search algorithm is also a tree searching algorithm and a backtracking algorithm that is typically used to generate the possible game states. This algorithm finds the optimal move for a player, assuming that the opponent also plays optimally (Nasa et al., 2018). The technique that can be used to optimize the MIN/MAX algorithm is the application of $\alpha\beta$ pruning. When MIN/MAX with $\alpha\beta$ pruning is used instead of a simple MIN/MAX algorithm, less number of possible nodes is evaluated in the game tree.

The $\alpha\beta$ pruning is not altogether a different algorithm than MIN/MAX; rather it is an optimized version of the algorithm that applies evaluation function to each leaf node in the game tree and selects the node with the highest evaluation based on the MIN/MAX principle (Kato et al., 2015; Nasa et al., 2018). Another perspective of the algorithm can be directed towards the strategy of the MIN/MAX tree search pruning which is best-first $\alpha\beta$ pruning and depth-first $\alpha\beta$ pruning (Plaat et al., 2015). This study highlights one important question; the possibility of formulating the best-first search procedures using depth-first procedures. This concept had been formalized in a search indicator algorithm such as conspiracy number search (CNS) and its variants such as proof number search (PNS) and single conspiracy number (SCN) search.

Conspiracy Number Search (CNS) is a MIN/MAX search tree algorithm that tries to measure the possibility of a root node changing its value in the MIN/MAX search tree (McAllester, 1988; Schaeffer, 1990; Plaat et al., 1995). The algorithm does not guarantee that the correct solution will be found when it terminates, but the most likely one instead by employing a probabilistic search (Khalid et al., 2015b). The conceptual framework behind the CNS is to measure the search trees growth with the conspiracy numbers (CN) to have the highest confidence. The search is guided in a best first manner, where the tree searched so far is kept in memory. A recent formalism of the CNS can be found in (Vu et al., 2016).

Another well-known search indicator is the proof number search (PNS). The PNS is best-first tree search algorithm in the AND/OR searches tree (Allis et al., 1994). PNS focuses on an AND/OR tree and tries to establish the game-theoretical value in an efficient, greedy least-work-first manner (Ishitobi et al., 2013). The AND/OR tree is a type of game tree where the nodes have only three possible values: true, false and unknown. The main idea is to assign a proof number (PN) and a disproof number (DN) to each node of the partially searched AND/OR tree. A PN shows the number of nodes required to prove the node while a DN shows the opposite. The PNS algorithm will choose the most-proving node to guide node expansion where it can effectively exploit narrow and deep branches that seem to be promising. They are particularly effective in games where the branching factor is non-uniform (Gao et al., 2017). A recent formalism of the PNS can be found in (Ishitobi et al., 2013).

Nevertheless, CNS requires significant computing power and memory. Although PNS reduces the memory footprint required by CN into two factors (PN and DN), the PNS application is limited to the framework of the AND/OR tree search. In recent years, CNS is gaining new grounds where it had been utilized to identify critical positions in a speculative play (Khalid et al., 2015b; Khalid et al., 2015a) and improve move selection (Vu et al., 2016). While PNS had been independently adopted and improved (such as the work in (Ishitobi et al., 2013) and (Ishitobi et al., 2015)), a new concept emerged from both CNS and PNS, called single conspiracy number (SCN), had been introduced in the recent years.

Single conspiracy number (SCN) involves using an indicator that trying to estimate the difficulty of the current game state to get a value more than a predefined threshold in the MIN/MAX tree search (Song and Iida, 2017). SCN was defined as an intermediate of conspiracy number and proof number, which indicates the difficulty of a root node changing its MIN/MAX value to a certain score decided by a threshold. SCN was recently proposed to analyze the game progress patterns (Song and Iida, 2017) which was applied to the Chinese Chess. In this paper, SCN is applied to another two-players perfect-information game called Checkers where the SCN is calculated in multiple situations and their effect in each situation is determined. Although Checkers had been solved previously by Schaeffer in 2007 (Schaeffer et al., 2007), its considerably large search space provides a good ground for SCN.

3 SINGLE CONSPIRACY NUMBER IN CHECKERS

Checkers is a two players perfect information board game played on square checkers board consists of 64 light and dark squares. Each player has 12 playing disk-shaped pieces (Singh and Deep, 2014). The game starts by having 12 pieces (red and white) arranged on the board for each player (Figure 1). Red moves first and players take their turn by advancing a piece diagonally forward to an adjoining vacant square. If an opponent's piece is in an adjoining diagonally vacant square, with a vacant space beyond, it must be captured and removed by diagonally jumping over it to the empty square. If this square presents the same opportunity, successive jumps forward in a straight or zigzag direction must be taken; so-called "forced-jump" (Al-Khateeb and Kendall, 2012). If multiple forced jumps are available, the player may choose which one to make. If a piece reaches the king row (last row) then it can move backward also. A player will win when all the opponent's pieces are captured or blocked. A game is declared a draw when neither side can force a victory nor the trend of play becomes repetitive.

In the following subsections, the application of SCN concepts to the checkers is briefly described in Section 3.1. Then, the experimental design and setup is discussed with respect to the SCN concepts applied to the open checkers source software called "Samuel" (Section 3.2).

3.1 Application of Single Conspiracy Number

The positive MIN/MAX value shows the advantage of the WHITE player while the negative number shows the advantage of the RED player. Let *n.scn* be the SCN of a node *n* and *m* be the MIN/MAX value of node *n*. *T* is a threshold of the legal MIN/MAX values. The formalism of the single conspiracy number is given as follows.

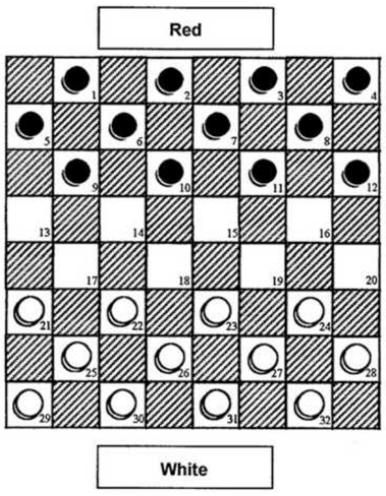

Figure 1: Checker board (adapted from (Singh and Deep, 2014)).

- When n is a terminal node
  1. If $m \geq T$, $n.scn = 0$
  2. If $m < T$, $n.scn = \infty$
- When n is a leaf node (not terminal)
  1. If $m \geq T$, $n.scn = 0$
  2. If $m < T$, $n.scn = 1$
- When n is an internal node
  1. If n is a MAX node: $\min\limits_{n_c \in \text{child of} n} n_c.scn$
  2. If n is a MIN node: $\sum\limits_{n_c \in \text{child of} n} n_c.scn$

The measurement of the result was conducted by recording the value of SCN based on the search depths d of the game state. The result was observed using the sliding window that calculates the SCN values with different consecutive d where the initial $d = 2$. The variation of SCN was also observed as the game progresses to determine its stability from the beginning to the end.

3.2 Experiment Design and Setup

To conduct the experiment, a popular open source checkers project called "Samuel" was used as the base software where the SCN was implemented on top of the original search engine of Samuel. The original search engine of Samuel implements αβ search and various techniques such as transposition table and quiescence search to speed up the search process. The experiment was also conducted by making two Samuel's search engine compete in different matches against each other. In the leaf nodes, a node value from the heuristic function, which is the quiescent search, was preserved within Samuel and directly applied to the SCN calculation. Meanwhile, the transposition table within Samuel is switched in and out during the experiment to observe its effect on the SCN values.

The SCN threshold was set to 150, 300 and 1900 from the range of $[0, 2000]$. These values were set differently in the experiment to observe changes in the SCN values when the threshold changed. Winning, drawing and losing positions were considered and discussed separately to analyze the resulting SCN changes in each situation. The SCN value was calculated on each node in the αβ search tree. A single conspiracy number measures the difficulty of the current node (current player) getting a heuristic value over a specified threshold. Only even d value was considered as a complete block of MIN/MAX search tree. This experiment was conducted on a search tree with $d \in [2, 16]$. As such, based on the three aforementioned threshold with eight different search tree depths, a total of 24 matches were simulated.

In some cases, Samuel's AI returns a specific value to show pruning status like "timeout" or "bad nodes" in the αβ search. The value of those nodes with special meaning values was reevaluated with the original board evaluation function within Samuel's implementation. The heuristic function implemented in Samuel will be positive if the WHITE player is having an advantage and will be negative if the game is in a favor of the RED player side.

Iterative deepening was originally deployed with the αβ search to get next play properly, even if some branches within a search tree got cut out when the time is up. Since the experiment's goal to observe the differences of SCN values in a search tree with different d, the SCN were calculated separately for each d during the processing of the iterative deepening algorithm. In rounds that some high d value ($d \geq 17$), that d are neglected and the SCN for the uncalculated d will be set as "UNINITIALIZED".

4 RESULTS AND DISCUSSION

At the beginning of the game, the SCN values in each d were varied since the board at the beginning position does not show any advantages for each side of the players. So, each player is likely to get a score lower than a threshold $T = 150$. For d that are not deep enough to reach a state in which the computed heuristic score is not more than T, SCN values will be very scattered depending on the number of nodes explored

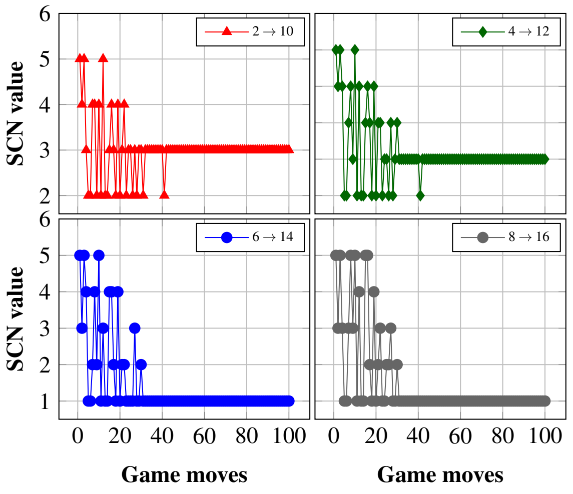

Figure 2: The variation of SCN values for WHITE player (drawing) for variation of search depth (d).

by the algorithm. As the game progresses, moves that obtained heuristic score over the threshold T will be found which affect the SCNs. Thus, the SCN values are different in winning, drawing and losing position for each player.

In a draw game, the SCN values switched back and forth depicting looping scenario where no player can have too large advantage over the other. As shown in Figure 2, the interval of higher d started becoming less varied and at $d = 8$ to $d = 16$, the SCNs become very similar. Also, it can be seen clearly that SCNs of the search $d = 2$ and $d = 4$ are different to the others. Figure 3 and Figure 4 shows that the SCNs in higher d interval are likely to be more stable in both winning (WHITE player) and losing (RED player) situations as the values in those interval are less varied. The SCNs from each interval were nearly constant in the late game because the game was struck in a "loop" play. The SCN values from the loop cycle were slightly varied due to the different plays found during the search. However, as the search got deeper, the SCN values were converging to the same value.

The SCN values normally become near 0 in the winning positions and become ∞ in a losing position. However, this value sometimes may not retains its stability. Even in the case that the evaluation was 0 where the position indicates the player has a very large advantage over its enemy, there were still some moves that may lead to worse positions that were neglected in the previous search tree. Similarly, there was also a move that was evaluated as ∞ (the player is in the losing position) that still found moves that may lead to a position with a value over the threshold.

From the perspective of the entire gameplay with various d values (Figure 5), the SCN values of both

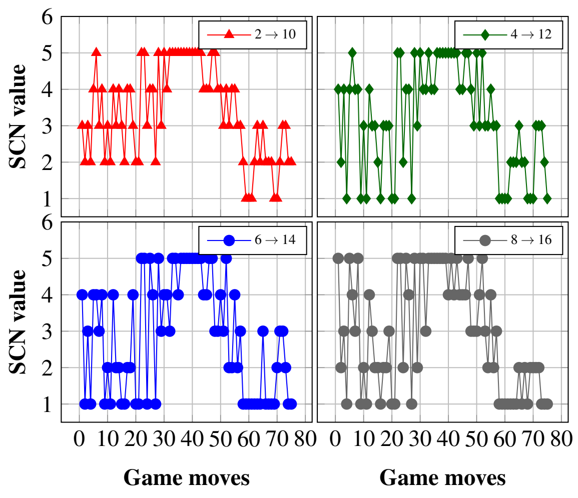

Figure 3: The variation of SCN values for WHITE player (winning) for variation of search depth (d).

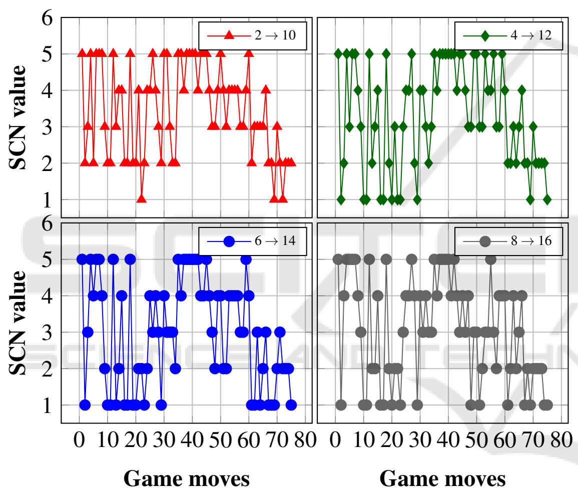

Figure 4: The variation of SCN of red player (losing) for variation of search depth (d).

players will get very unstable in the middle stage of the game where the WHITE player is getting an advantage against the RED player over the threshold $T = 150$. The SCN values will then become stable again and the advantage of one player does not easily changed (WHITE player obtained the heuristic score more than the threshold). Compared to the early stage of the game, none of the players get advantage over one another since their heuristic score is no larger than the threshold $T = 150$.

When the endgame is a draw (Figure 6), the SCN values for most of the time after the 42^{nd} moves stabilizes all the value of 3. This means $d \in [6,16]$ gets the same SCN values in a draw game while each player gets their own advantage back and forth over a few moves before the 42^{nd}. Based on this, it can be hypothesized that there is a move that can "dominate"

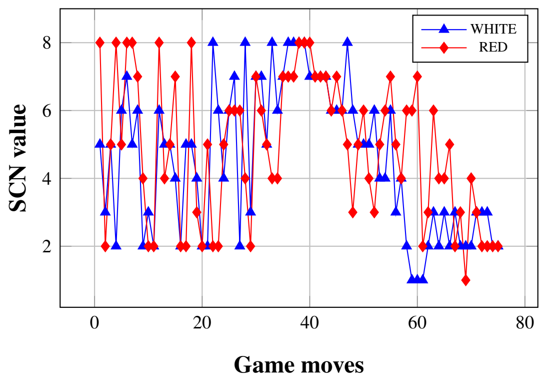

Figure 5: Player's moves number against different values of SCNs calculated for $d \in [2,16]$ (8 different even d).

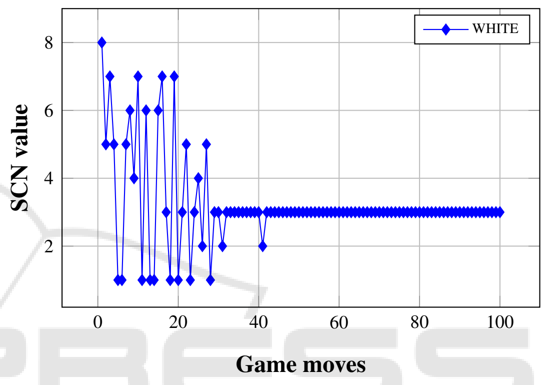

Figure 6: SCN variation of the WHITE player during the game progress for a draw game.

other moves in the search tree and this move has been found. It is worth to mentions that the SCN values for the search with $d > 6$ of that dominating move are all the same. In addition, the experiment calculated the SCN values in the $\alpha\beta$ tree search process, but the decision was not made based on the SCN values. Although the dominating moves could not be found when the $d = 2$ and $d = 4$, it could have existed within those d at some point as the game progress.

Another similar experiment was conducted with the transposition table removed. The number of nodes explored by the tree search algorithm will expand enormously since checkers is a game that has only two playable pieces for each player. As such, in this experiment, the threshold of the SCN was set to $T = 300$ where the focus of the experiment was on the states of one player gets a major advantage by a large margin. Since the search process is slower without a transposition table, the experiment was observed for $d \leq 14$.

As shown in Figure 7, the SCNs of the winning RED player were grouped by three value of d. It can be observed that the variation of SCN values for higher d is likely to be less than those of lower d

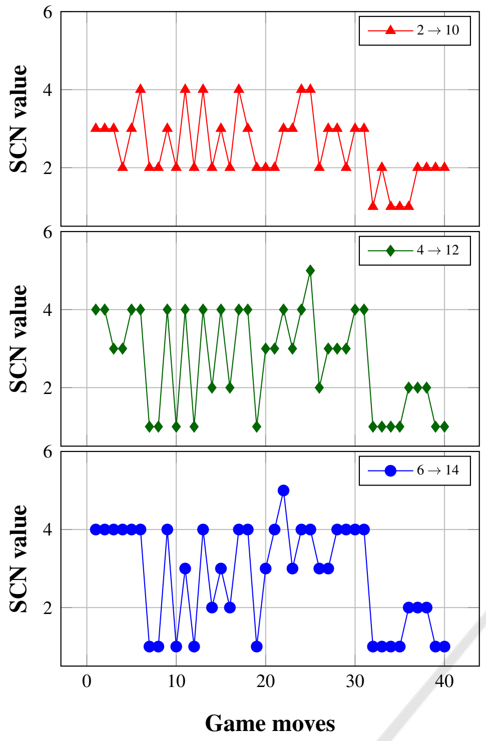

Figure 7: Red SCN (winning player) with the SCN threshold of $T = 300$.

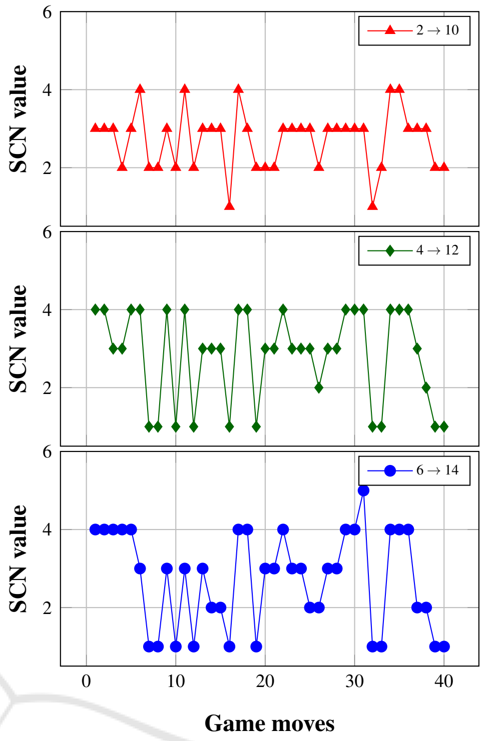

Figure 8: Red SCN (winning player) with the SCN threshold of $T = 1900$.

throughout the progress of the game. This means the computed SCN will be more stable in higher d value even when the transposition table was removed and the number of explored nodes were expanded. Subsequently, the SCN threshold was changed to $T = 1900$ which is very close to the maximum available heuristic value of 2000 in the Samuel's checkers' heuristic function. That means the only nodes leading to a win within a few moves will get SCN of 0 while the others will be counted as 1. As such, observing Figures 8, the result remain the same as every previous experiment. The SCN values for higher d are more stable with compared to the lower d even when the SCN threshold is getting very close to the maximum limit of the heuristic function.

The observation of the $d \in [2, 14]$ in a winning RED player with an SCN threshold of $T = 1900$ was shown in the Figure 9. In the middle stage of the game, the SCN values become unstable when a winning player (RED player) is going to get an advantage over the other player. On the other hand, when the SCN threshold is lower, the SCN values gain stability when d is high or when a game is going to end. This reinforces the observation made previously where d affects SCN values especially when the advantage of a player is over the threshold of T. Most of the time, the search technique like $\alpha\beta$ search is not implemented

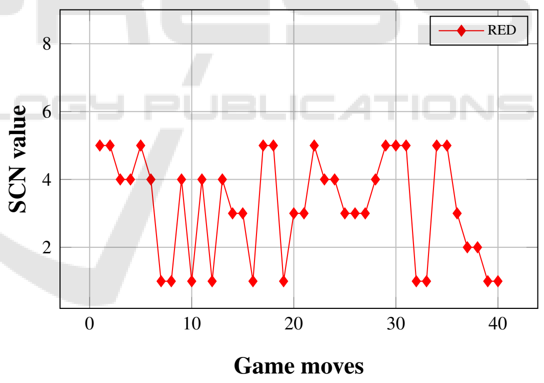

Figure 9: SCN with $d \in [2, 14]$ of the RED player during the game progress for a winning game with threshold $T = 1900$.

alone. As in Samuel, transposition table and quiescent search are also implemented with a number of minor optimization technique. SCN value got affected greatly when the search tree is pruned too much and the search algorithm explored only a few nodes. This can result in all SCNs in an entire game become 0 or 1.

This situation can be fixed by increasing the value of d (search depth). Figure 10 shows SCN values calculated for $d \in [2, 16]$ where the original implementation of Samuel is used. The SCN values for $d = 2$ unable to indicate the progress of the game since only

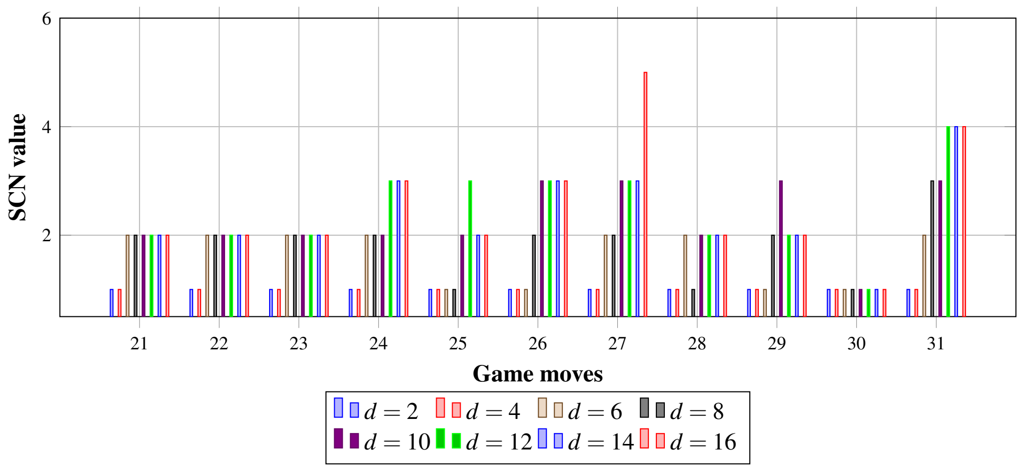

Figure 10: SCN of WHITE player for $d \in [2,16]$ with threshold of $T = 300$ for move 21st to 31st.

one MIN/MAX level was calculated and the root node will get the SCN value directly from one of its children. For $d = 4$, the SCN values are still all 1 because only few nodes per d level were explored. Meanwhile, the SCN value of the root node has a wider range of values when the value of d increased. This is because, even if the tree is very unbalanced, more nodes will be evaluated at the bottom of the tree and the range of possible values for SCN in each tree level increases.

The SCN works very well when the search tree is balanced since each node will have the same possible SCN values. However, in an ideal case that an evaluation function has very high accuracy, the tree is likely to be unbalanced where only few child nodes were explored at each node. In checkers, there is a "forced-jump" rule which forced a player to capture the opponent's piece when it is possible. This makes some nodes in the search tree has only one child node and those nodes will have a very narrow range of SCN values (0 or 1). A forced-jump rule in checkers forms the basis of all tactics in checkers, thus causing the search tree in checkers to be unbalanced in every game.

5 CONCLUSION

In this paper, the single conspiracy number (SCN) were calculated for each move made for the entire game progresses considering various search depth (d). Comparison of different SCN values was calculated from a different d values which measure the frequency of SCN values of the root node changing its own value. This paper found that SCN value for a winning position is low (approaching 0) while the SCN value of a losing position is high (or approaching ∞) if there is no way to win the game from that position.

In an ideal case that the heuristic evaluation is accurate, the shape of the tree is likely to be unbalanced and the calculated SCN values might not be usable. This is because of the rule called "forced-jump" in checkers where many nodes in the search tree were compressed into a single child and make the SCN value of every state in the game become only 0 or 1. Since the SCN is usually calculated online with the search algorithm, its value may also be affected by the shape of the search tree. In a very unbalanced tree where the number of the child node is varied, the SCN values will become unstable.

For low value of d, the SCN could get affected greatly by the shape of the tree and become unstable since every node reachable from the current state were evaluated with the score lower than that of the threshold T. As the search gets deeper, the likeliness of getting the reachable state with the score more than the threshold T also increases, thus enabling the SCN value to classify good and bad states of the game.

However, the SCN only focuses on measuring the difficulty of a node getting an evaluated score over the threshold of T. In addition, the potential areas of SCN as a measure of difficulty in a decision-making process had not been studied. Further works that combine the SCN with other heuristic evaluation function in order to get a better search performance provide grounds for future work.

ACKNOWLEDGEMENTS

This work was carried out on a transfer scheme between Japan Advanced Institute of Science and Technology (JAIST) and Chulalongkorn University (Thailand) Internship Program in 2018. This research is funded by a grant from the Japan Society for the Promotion of Science, within the framework of the Grant-in-Aid for Challenging Exploratory Research.

REFERENCES

Al-Khateeb, B. and Kendall, G. (2012). Effect of look-ahead depth in evolutionary checkers. *Journal of Computer Science and Technology*, 27(5):996–1006.

Allis, L., van der Meulen, M., and van den Herik, H. (1994). Proof-number search. *Artificial Intelligence*, 66(1):91 – 124.

Gao, C., Müller, M., and Hayward, R. (2017). Focused depth-first proof number search using convolutional neural networks for the game of hex. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 3668–3674.

Ishitobi, T., Cincotti, A., and Iida, H. (2013). Shape-keeping technique and its application to checkmate problem composition. *Artificial Intelligence and Game Aesthetics: the 2013 AIIDE Workshop (WS-13-19)*.

Ishitobi, T., Plaat, A., Iida, H., and van den Herik, J. (2015). Reducing the seesaw effect with deep proof-number search. In *Advances in Computer Games*, pages 185–197. Springer.

Kato, H., Fazekas, S. Z., Takaya, M., and Yamamura, A. (2015). Comparative study of monte-carlo tree search and alpha-beta pruning in amazons. In *Information and Communication Technology*, pages 139–148. Springer.

Khalid, M. N. A., Ang, E. M., Yusof, U. K., Iida, H., and Ishitobi, T. (2015a). Identifying critical positions based on conspiracy numbers. In *International Conference on Agents and Artificial Intelligence*, pages 100–127. Springer.

Khalid, M. N. A., Yusof, U. K., Iida, H., and Ishitobi, T. (2015b). Critical position identification in games and its application to speculative play. *Proceedings of the 7th International Conference on Agents and Artificial Intelligence (ICAART 2015)*.

McAllester, D. A. (1988). Conspiracy numbers for min-max search. *Artificial Intelligence*, 35(3):287–310.

Nasa, R., Didwania, R., Maji, S., and Kumar, V. (2018). Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game. *International Research Journal of Engineering and Technology (IRJET)*, 5(4):1637–1641.

Plaat, A., Schaeffer, J., Pijls, W., and De Bruin, A. (1995). Best-first fixed-depth game-tree search in practice. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 273–279, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Plaat, A., Schaeffer, J., Pijls, W., and de Bruin, A. (2015). Best-first and depth-first minimax search in practice. *arXiv preprint arXiv:1505.01603*.

Schaeffer, J. (1990). Conspiracy numbers. *Artificial Intelligence*, 43(1):67 – 84.

Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007). Checkers is solved. *science*, 317(5844):1518–1522.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Singh, A. and Deep, K. (2014). Use of evolutionary algorithms to play the game of checkers: Historical developments, challenges and future prospects. In *Proceedings of the Third International Conference on Soft Computing for Problem Solving*, pages 55–62. Springer.

Song, Z. and Iida, H. (2017). Using single conspiracy number to analyze game progress patterns. In *Computer, Information and Telecommunication Systems (CITS), 2017 International Conference on*, pages 219–222. IEEE.

Vu, Q., Ishitobi, T., Terrillon, J.-C., and Iida, H. (2016). Using conspiracy numbers for improving move selectionin minimax game-tree search.