A Gomoku Game-Testbed for Monte-Carlo Tree Search Algorithms

Lisa Liu[1][0009−0007−8259−6408] and Kelvin Yu[2]

[1] University of California, San Diego, La Jolla CA 92093, USA
lil043@ucsd.edu
[2] Oxford, Oxfordshire, England
kelvin2yu@gmail.com

Abstract. Gomoku is a strategy game played on the Go board. Two players, black and white, alternate placing pieces on the intersections of the grid to form five-in-a-row. It has often been used to test tree search algorithms, including Monte-Carlo Tree Search (MCTS). Due to its simplicity, it is possible to introduce variations to the rules slightly without affecting the core mechanics and strategies for the game. In this paper, we focus on the board size (traditionally 15x15) and introducing a dynamic boundary for moves (traditionally unbounded). Both of these qualities greatly change the size of each move's action space. We contribute 9 variants of Gomoku with different combinations of board sizes and dynamic boundaries, in order to provide a set of settings ranging across action space sizes. We calculated the action space sizes per moves for each board, and implemented a new variant of MCTS that uses ancestor-based alpha-beta bounds in the selection phase. We ran this against the classical MCTS in order to demonstrate the effects of the action space changes.

Keywords: Gomoku · Monte Carlo Tree Search · Action Space

1 Introduction

Gomoku, also called "Five in a Row," is a two-player strategy board game. It is zero-sum and is a game with complete information. It's traditionally played on a Go board, and its objective is straightforward; form a straight line of five pieces in a row. There are many variations on its rules, such as not allowing six pieces in a row, or an "overline," to count as a win, or implementing various swapping rules in order to mitigate the first player advantage. Due to its simplicity and adaptability, Gomoku has often been used as a subject of research in the past, dating back to the 1993, with a paper on threat space search, a type of tree search [1]. Subsequent research on tree searches were also tested on Gomoku [2], and it is still a subject of analysis over 20 years later [8].

In this paper, we will seek to design different variations of this game in order to test out the effects of different board types on tree search algorithms. In particular, we want to focus on Monte-Carlo Tree Search (MCTS), and will demonstrate the usage of our testbed game with a new MCTS variant from 2024, which is explained in the next section.

2 Ancestor-Based Alpha-Beta Bounds for Monte Carlo Tree Search

2.1 Alpha-Beta Pruning

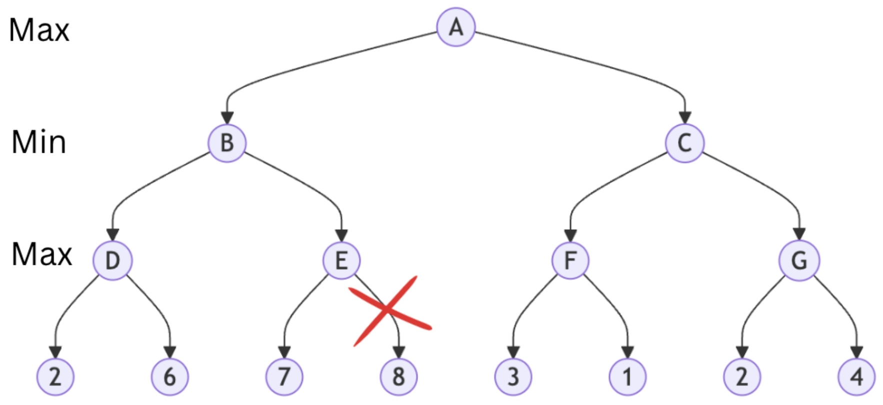

Fig. 1. Example to illustrate alpha-beta pruning on a minimax trees.

MCTS [5,9] is a tree search technique that uses random sampling and statistical evaluation to explore only the most promising areas of the search space, rather than exhaustively trying every possibility. The algorithm has four phases: 1. selection, 2. expansion, 3. simulation, and 4. back-propagation. Every node contains a current state; for Gomoku, it is the board, the placed pieces, and which player's turn it is. Its children nodes are states that are reachable in one action. The selection phase chooses the child node to explore next, either picking an untried action, or the child that returns the greatest value using the upper confidence bound (UCB) formula. In the expansion phase, all the possible actions and board states from the selected child node are added to the tree. In the simulation phase, moves are placed until the result, win or loss, is determined (a "rollout"). In back-propagation, this result is recorded up the tree back to the root node, so the root node is aware of whether this child node seems more or less promising to explore in the future. In this paper, we are focusing on the selection phase, and we use random rollouts (random moves until the end).

Alpha-beta pruning is an optimisation technique used in primarily for minimax algorithm [4]. Figure 1 shows the core concept. We start at node A and look at its first child, node B. We then look at node D. Here, the maximising player will choose the value 6 because it is the greatest, and this gets returned to B. To the minimising player here, choosing node D would result in a value of 6 so they are already guaranteed a value of 6 or less. Now we look at node E. Its first child has a value of 7, so we know the max player is guaranteed a value of 7 or more if the min player chooses node E. Because of this, the min player would never even consider node E because they can get a better value of 6 at node D. This means we did not have to look at the remaining child of node E because it would not affect the decision of the min player. In this example, only one node was pruned, but in larger trees, entire branches can be ignored which saves a lot of computational time.

When implemented, a variable "alpha" keeps track of the minimum value that the max player is guaranteed, and likewise "beta" represents the maximum value that the min player is guaranteed.

2.2 Ancestor-Based Bounds

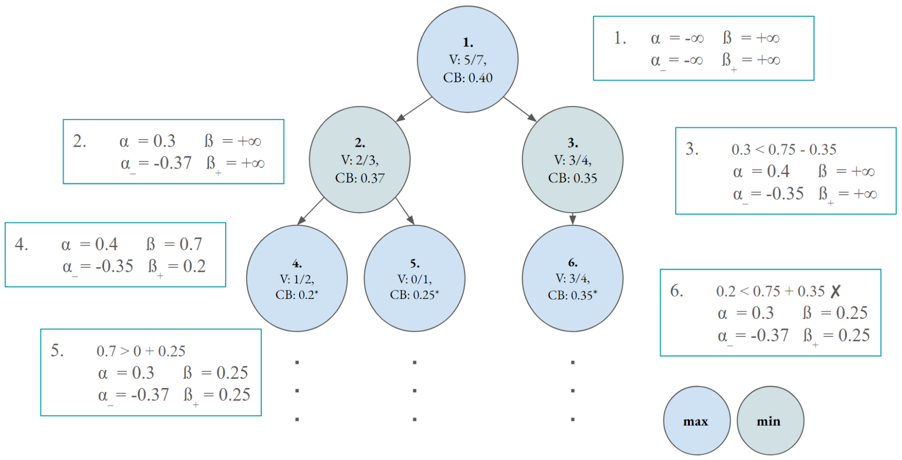
1. V: 5/7, CB: 0.40
2. V: 2/3, CB: 0.37
3. V: 3/4, CB: 0.35
4. V: 1/2, CB: 0.2*
5. V: 0/1, CB: 0.25*
6. V: 3/4, CB: 0.35*
1. α = -∞ ß = +∞ α_ = -∞ ß+ = +∞
2. α = 0.3 ß = +∞ α_ = -0.37 ß+ = +∞
3. 0.3 < 0.75 - 0.35 α = 0.4 ß = +∞ α_ = -0.35 ß+ = +∞
4. α = 0.4 ß = 0.7 α_ = -0.35 ß+ = 0.2
5. 0.7 > 0 + 0.25 α = 0.3 ß = 0.25 α_ = -0.37 ß+ = 0.25
6. 0.2 < 0.75 + 0.35 ✗ α = 0.3 ß = 0.25 α_ = -0.37 ß+ = 0.25
max
min

Fig. 2. Example to illustrate alpha-beta pruning on a MCTS tree. Value (V) is the number of wins / number of visits per node, while confidence bound (CB) is the c * sqrt(ln(parent visits) / node visits). We used c = 0.5. The ... represents the unexplored future moves.

In Pepels and Winands' 2024 paper [7], they describe a way to use alpha-beta pruning with MCTS. Alpha-beta pruning is used for minimax trees because minimax is a deterministic algorithm and each state will always have the same value, while MCTS relies on random rollouts so depending on the amount of rollouts so far, a state's value might change [3]. *Alpha bounds* refer to the lower bound of the values that the maximising player is guaranteed, regardless of what the minimising player does. *Beta bounds* are the opposite, being the upper bound of the value that the min player is guaranteed. Because of MCTS's random rollouts, the true alpha or beta bound might not be found. However, the ancestor node in the tree, given enough rollouts, may hope to keep a running global alpha and beta bound that can still be used to eliminate choices, even if it is not the true alpha and beta value.

Figure 2 shows the effect of alpha-beta pruning on MCTS. Each node has a value

$$V = \frac{\text{number of wins}}{\text{number of visits}}$$

and a confidence bound

$$CB = C \times \sqrt{\frac{\ln(\text{number of visits to parent nodes})}{\text{number of node visits}}}$$

$V + CB = UCB$ value of the node. The global alpha-beta bounds, alpha-minus and beta-plus, are initialized to negative infinity and positive infinity respectively, and each time, depending on whether the node is minimizing or maximizing, either the alpha-minus or beta-plus will be updated if the previous alpha or beta value is less than or greater than the lower or upper bound of the value, respectively. This way, an alpha and beta bound can be maintained even without trying all the possibilities.

In their paper [7], the Gomoku set-up was a 15x15 board with no dynamic boundary box. It started off with an empty board, and implemented the *pie rule*: the first player makes their move, and then the second player is free to swap with the first player's position, or make their own move. The results of the paper's Gomoku experiment indicated that ancestor-based alpha-beta bounds alone do not impact the performance of MCTS significantly, and it was hypothesized that this was due to Gomoku's wide-to-narrow branching factor. Amazons, the other game in the paper that showed little improvement, also shares the same branching factor scaling traits. It was noted that with a higher iteration budget, the improvement is more pronounced, potentially indicating that with a smaller iteration budget and a larger initial action space, the ancestor-based alpha-beta may not have been activated enough to have an impact [7].

To test out this hypothesis, we decided to test out ancestor-based alpha-beta bounds MCTS on various board set-ups. Changing the initial board size affects the initial action space size. If the theory about the alpha-beta bounds not activating is true, then with the same iteration budget, the bot should do better on smaller board sizes and worse on larger board sizes. Introducing a dynamic boundary box to the game flips the branch factor scaling, from *wide-to-narrow* to *narrow-to-wide*. The boundary box increases in size as time goes on, and although it is possible to make moves without increasing the boundary box size, each expansion increases the action space size significantly, until there is no more space on the board to increase the boundary box. If the wide-to-narrow branching factor caused the ineffectiveness in the original experiment, then we should be able to see more effective results with bounding boxes.

We implemented a bot using this method, which will hereafter be referred to as alpha-beta MCTS.

3 Gomoku Settings

3.1 Board Set-Ups

For board size, we tested the following sizes: 11x11, 15x15, 19x19. 19x19 is the traditional board size, shared with the game of Go. 15x15 is the most commonly used board size today, as well as the size used in the ancestor-based alpha-beta bounds paper [7]. 11x11 is a board size we chose in order to provide an even smaller board for comparison.

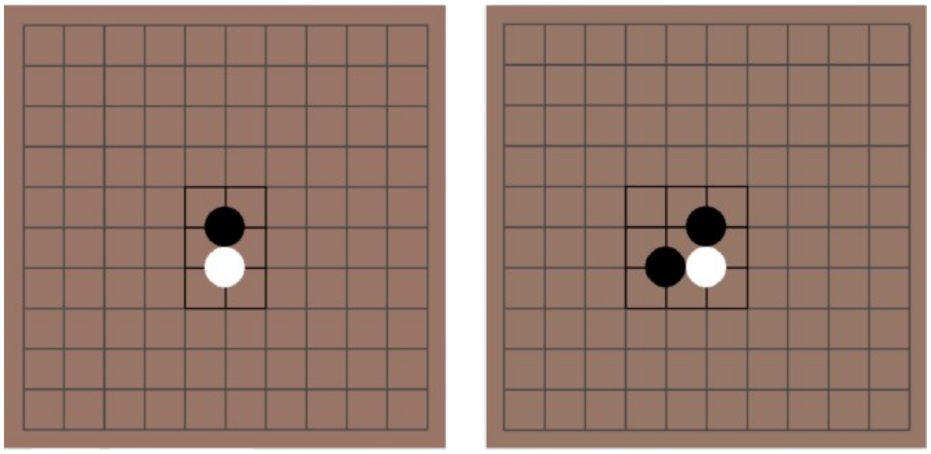

Fig. 3. Dynamic boundary expansion, 11x11 with radius 1.

We test dynamic boundary boxes with radius 1, 2, and infinity. Radius 1 means that the boundary is always 1 grid away from the all the current pieces on the board, unless the boundary is already on the edge of the board. Figure 3 illustrates how this boundary expands dynamically with more pieces placed. Radius 2 means that the boundary is always 2 grids away from the existing pieces on the board. Infinity means that there is no boundary box; this is the usual Gomoku version, where pieces can be placed anywhere on the board from the start.

In total, we have 9 board set-ups: every combination of the 3 board size settings (11x11, 15x15, 19x19) and 3 dynamic boundary settings (radius 1, 2, infinity).

Because we wanted to design the boards to be as simple as possible in order to focus on the results of changing the branching factor and the size of the action space as the game progresses, many of these boards would necessitate a change in the usual strategies used to play. For example, many of the board configurations that are displayed in a 1993 paper analyzing human experts and threat space search require pieces to be played in the corners and edges early on [1], which is not possible with dynamic bounding boxes of a small radius. A smaller 11x11 board may also make it harder to execute some of the strategies by reducing the amount of space that can be inserted between pieces. More research would have to be done on seeing how human players have to adapt to the changes in these rules.

3.2 Action Space Size Calculations

In order to simplify the game, we removed the pie rule and instead ensured fairness when testing both the alpha-beta MCTS and regular MCTS against each other by running an equal number of simulations where each bot goes first as black. For simplicity and to ensure greater variability of games with a limited training budget, we also initialized the board with black having a piece in the center and white having a piece placed randomly within the boundary of the black piece. The first piece in the center was to ensure consistency for the dynamic boundary boxes. Starting with a piece in the corner, for example, would greatly change the size of the bounding box, as two of the box's sides are frozen from the start and unable to expand. By ensuring the boxes are centered at the beginning, we set the game up to allow the boundary boxes to grow as evenly as possible throughout the game.

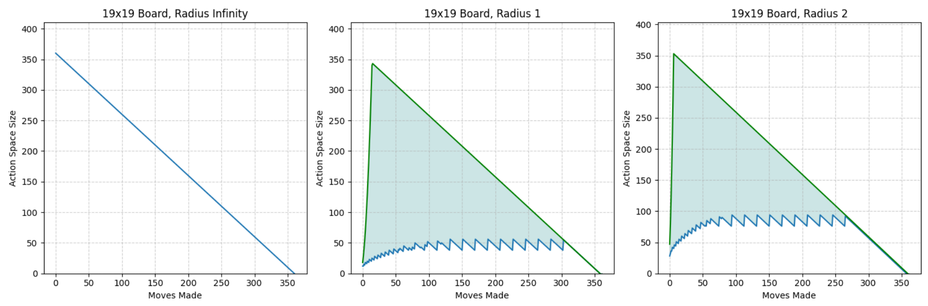

Fig. 4. Action space progression of all 3 boundary radii, for a 19x19 board. Green is the upper bound. Blue is the lower bound. The shaded area covers all the possible action spaces at that move.

The leftmost graph in Figure 4 shows the action space sizes changing across moves for a 19x19 board with a boundary of radius infinity, or no boundary. In the 11x11 board for example, 2 of the 121 grid intersections are filled, so at move 1 (black), 119 actions are available. By move 2 (white), only 118 actions are available, and so on. The graph displays the upper bound for the action space sizes, not accounting for wins/losses before the whole board is filled. For the 15x15 and 19x19 boards, the only difference is that the starting number of actions is 223 and 359, respectively.

The center graph in Figure 4 shows the action space sizes changing across moves for a 19x19 board with a boundary of radius 1. The upper bound was calculated through adding a piece in the corner of the box, ensuring maximum expansion per move, until the boundary covers the entire board, after which it shrinks along with the infinity radius graph. The lower bound was calculated taking the minimum at each step of multiple combinations of picking where to expand (side vs corner) and then always filling in the box completely before expanding again. All the possible action space size progressions are contained between these two bounds. The upper bound represents a narrow-to-wide branching factor before becoming wide-to-narrow like the unbounded Gomoku, while the lower bound represents an overall consistent branching factor, albeit with an action space size much smaller than the unbounded Gomoku. However, most games (games within the bounds) generally follow a narrow-to-wide branching factor until the board is fully expanded.

The rightmost graph in Figure 4 shows the action space sizes for a 19x19 board with a boundary of radius 2. It is very similar to the graph for radius 1, but the entire board is covered in fewer moves, so it hits the radius infinity line sooner. The games of this type can also have a narrow-to-wide branching factor, although the action space sizes will likely be larger than that of radius 1 at most steps.

These graphs were generated for a 19x19 board, but the 11x11 and 15x15 boards follow nearly identical trends, only with the maximum spaces on the board being 121 and 225 respectively, rather than 361.

4 Results

This section presents the results of 9 experiments on game boards of three different sizes.

4.1 Selection Phase

In the selection phase of MCTS, a node is picked for rollout. This node must be a terminal node (leaf node). Starting from the root node, the next node to traverse to is selected either for expansion (expanding the tree with all child nodes for the selected nodes, since it is a new node with no prior visits), with the regular UCT formula for selecting the best child (upper confidence bound applied to trees [6]), or the modified UCT formula that uses alpha-beta bounds to select the best child. We used "AB" to represent the total number of times the alpha-beta UCT was used, with "Final AB" being the subset where the final leaf node that got a rollout was chosen with alpha-beta UCT. "Reg" represented the times regular UCT was used, with "Final Reg" being the subset where the final child was chosen with regular UCT. "Expansions" represents the number of times the node chosen had no prior visits (neither UCT formula could be used).

In every selection iteration, both bots got a rollout budget of 5000. This means that for each move, expansions + final AB + final reg = 5000.

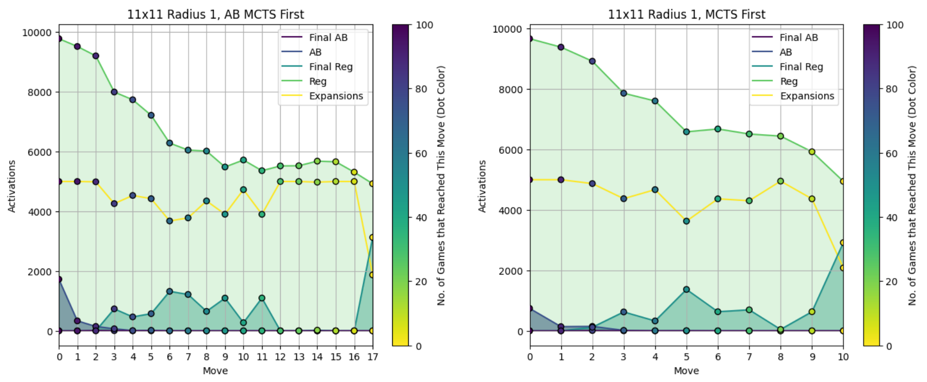

Fig. 5. Activations (separated by type) per move, 11x11 with radius 1.

Graphs generally follow the same trend for their radius type. Figure 5 shows that on an 11x11 board with radius 1, alpha-beta UCT does activate a noticeable amount, although only a very small, but non-zero, number of final selected nodes are chosen with alpha-beta UCT. However, as time passes, a lot of nodes are finally selected by regular UCT, rather than expansions, meaning that enough explorations have been done for the best child formulas to start activating. When MCTS goes first, a lot fewer moves are made by the AB MCTS, indicating it is either losing or winning faster. 15x15 and 19x19 radius 1's graphs also follow a similar trend. Radius 1 follows a narrow-to-wide branching factor that is low initially compared to the unbounded Gomoku games, and these results show that these configurations do allow for the initialization and usage of the alpha-beta bounds, even though it is not used a lot. Figure 6 shows the trend for dynamic boundaries of radius 2. The trend is similar to that of radius 1, but there are visibly less UCT activations of any kind throughout the selection phase as opposed to nodes selected for expansions. Unlike in Figure 5, where the number of final nodes selected for expansions dropped significantly throughout the game, with radius 2 there was a constant need to pick nodes for expansion throughout. This is likely due to the action space becoming too large after a few moves, creating a large influx of unexplored nodes that need to be expanded.

The upper bound of the 15x15 radius 2 action space size progression, assuming first two pieces are placed, goes as follows:

- (move 0 black, 23 spaces)
- (move 1 white, 46 spaces)

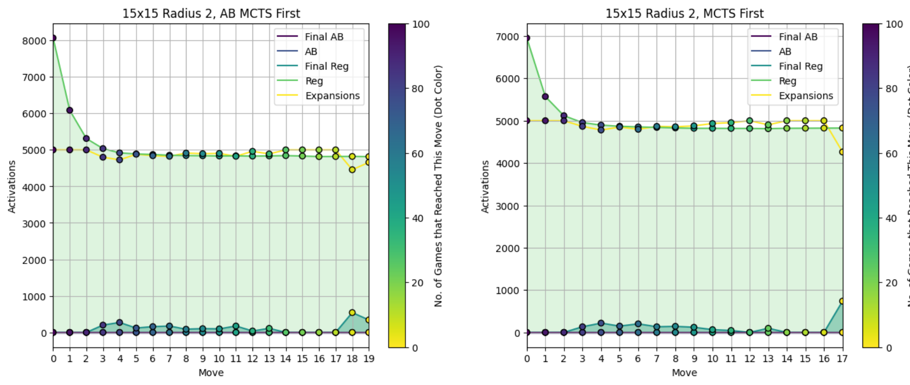

Fig. 6. Activations (separated by type) per move, 15x15 board radius 2.

– (move 2 black, 77 spaces)
– (move 3 white, 116 spaces)
– (move 4 black, 163 spaces)
– (move 5 white, 218 spaces)
– (move 6 black, 217 spaces)
– ...

The sudden jump from 163 spaces to 218 spaces on the 3rd black move coincides with the sudden drop of regular UCT activation around moves 2-3 for AB MCTS, indicating that UCT activation in the selection process does correlate with with the action space size. However, as soon as the action space stabilized, a small portion of final selections started to be made with regular UCT, as opposed to being entirely expansions. The ending spike in final regular UCT selections could be an outlier, since only 1-2 games out of 100 reached those last two games. Those games could be due to having a particularly favorable branching factor.

In Figure 7, the general trend for radius infinity is shown. Although both the radius 2 graphs and radius infinity graphs select expansions for almost all of their rollouts, regular UCT was used a lot less in the first few moves for radius infinity; in fact, regular UCT selection was consistent throughout the game. This is likely because the action space was at its largest throughout, unlike in radius 2, where the first few moves could work with a limited action space.

4.2 Win Rates

Table 1 shows the win rates from running alpha-beta MCTS and MCTS against each other 200 times per board, with 100 games where alpha-beta MCTS goes first and 100 games where regular MCTS goes first.

Alpha-beta MCTS showed a significant improvement compared to MCTS on the board with the smallest board with the tightest boundary, 11x11 radius 1, and did not show any improvement on the largest board with the largest

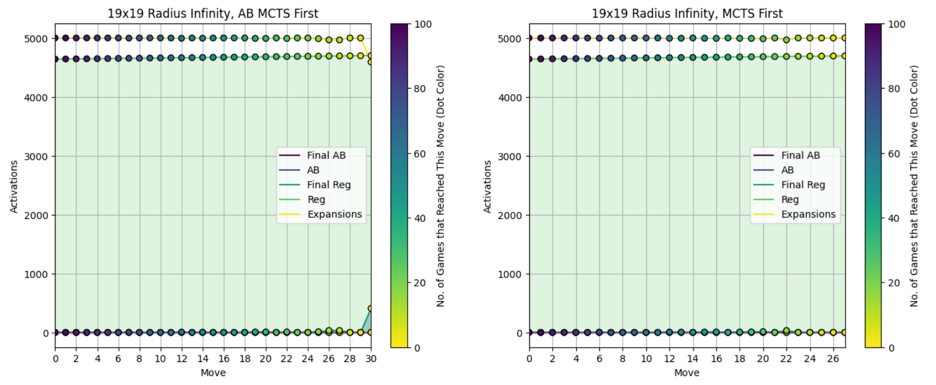

Fig. 7. Activations (separated by type) per move, 19x19 radius infinity.

Configuration	alpha-beta MCTS First	MCTS First
11x11 Radius 1	**74%**	53%
11x11 Radius 2	70%	56%
11x11 Radius ∞	67%	60%
15x15 Radius 1	57%	55%
15x15 Radius 2	68%	68%
15x15 Radius ∞	52%	44%
19x19 Radius 1	66%	48%
19x19 Radius 2	**80%**	60%
19x19 Radius ∞	50%	50%

Table 1. Comparison of alpha-beta MCTS and MCTS win rates as black across all board configurations. 100 games were run per board for each bot. Bold numbers are at least 20% higher.

boundary, 19x19 radius infinity. However, while lower radii and smaller board sizes seem to indicate more improvement generally, and the win rates do not always differ significantly. For example, 15x15 radius 1 only had a 2% win rate difference.

5 Conclusion

The alpha-beta MCTS algorithm we tested in this paper did show different results from the paper where the method was first described, which only used the 15x15 radius infinity board. 11x11 radius 1 in particular showed a significant win rate different (21%) with our rollout budget of 5000, showing that the branching factor of action space sizes can impact the results greatly even with a limited budget. Our experiments also give a concrete example of how the action space sizes can directly correlate with different features of MCTS, like the types of activations in the selection phase for alpha-beta MCTS.

The results of this paper help quantify how branching factors in games may affect the efficiency of different types of MCTS algorithms. While this is a useful tool to quantify the improvements made to MCTS, it can also be used help quantify the difficulty of the games in general, therefore helping game designers adjust the game playing experiences for all game players, AI or human.

Acknowledgments. We want to thank Tom Pepels, the author of "Ancestor-Based $\alpha - \beta$ Bounds for Monte-Carlo Tree Search," [7] who provided a lot of feedback on our implementation of his algorithm and insights on how to measure different qualities on its usage. We would also like to thank Dr. Sicun Gao, the professor of CSE 150B (Introduction to Artificial Intelligence: Search and Reasoning) at UC San Diego, for providing the original idea of a bounding box in Gomoku to limit the action space.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Allis, L., Herik, H., Huntjens, M.: Go-moku and threat-space search. Computational Intelligence **12** (10 1994)
2. Alus, L., van den Herik, 1, H., Huntjens, M.P.: Go-moku solved by new search techniques. Computational Intelligence **12**(1), 7–23 (1996)
3. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games **4**(1), 1–43 (2012)
4. Campbell, M.S., Marsland, T.A.: A comparison of minimax tree search algorithms. Artificial Intelligence **20**(4), 347–367 (1983)
5. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: International conference on computers and games. pp. 72–83. Springer (2006)
6. Kocsis, L., Szepesvári, C.: Bandit based monte-carlo planning. In: European conference on machine learning. pp. 282–293. Springer (2006)
7. Pepels, T., Winands, M.H.: Ancestor-based α-β bounds for monte-carlo tree search. In: 2024 IEEE Conference on Games (CoG). pp. 1–4. IEEE (2024)
8. Piazzo, L., Scarpiniti, M., Baccarelli, E.: Gomoku: analysis of the game and of the player wine. arXiv preprint arXiv:2111.01016 (2021)
9. Rimmel, A., Teytaud, O., Lee, C.S., Yen, S.J., Wang, M.H., Tsai, S.R.: Current frontiers in computer go. IEEE Transactions on Computational Intelligence and AI in Games **2**(4), 229–238 (2010)