Refined Monte-Carlo Tree Search Algorithm in Gomoku Game

Yuchuan Dong
yuchuan@chalmers.se

Zhiyuan Zhang
zhi-shan.zhang@connect.polyu.hk

Abstract

In the realm of artificial intelligence (AI) and gaming, the pursuit of developing sophisticated systems capable of challenging human intellect continues to captivate researchers and enthusiasts alike. Our project embarks on a compelling journey centered around the creation of an AI system dedicated to mastering Gomoku, or Five in a Row, a strategic board game played on a 15x15 grid. The algorithm we propose is based on Monte-Carlo Tree Search (MCTS). It is universally acknowledged that traversaling the whole Monte-Carlo Tree is impossible since the huge computation will cause a long time waiting. To achieve our goal in the context of Gomoku, we focus on two key aspects: distinctive heuristic knowledge and refined roll-out policies. Recognizing the unique strategy of Gomoku, we tailor the algorithm by incorporating game-specific heuristic knowledge. Additionally, our approach involves eliminating negative effects of the limitation of roll-out depth.

1 Introduction

The game of Gomoku has long been a fascinating challenge for both players and researchers in the field of AI. It is a two-player strategy game played on a square grid, where the objective is to be the first to form an unbroken row of five stones horizontally, vertically, or diagonally. AI algorithms have been developed to play Gomoku and compete against human players. Various strategies were employed. Theodoridis and Koutroumbas (2006) utilized pattern recognition , Kuhlmann and Stone (2006) tried heuristics, and tree search algorithms were analyzed by Chaslot et al. (2008) to make informed decisions and improve gameplay performance. Among these strategies, the Monte-Carlo Tree Search (MCTS) algorithm has shown great promise in tackling the challenges of Gomoku.

The primary objective of this project is to develop a refined MCTS algorithm specifically tailored for Gomoku, addressing the limitations of original MCTS algorithms and providing a more efficient and effective solution to playing the game optimally. The refined algorithm aims to improve decision-making, convergence speed, leading to superior gameplay performance.

The motivation behind this project is to make the MCTS algorithm applicable to the Gomoku game, with the intention of leaving room for future integration with deep learning algorithms, similar to those used in AlphaGo, to further enhance gameplay strategies. Gomoku presents unique challenges that require sophisticated algorithms to navigate the vast search space and make optimal decisions. By refining the MCTS algorithm, we aim to enhance its performance in Gomoku and contribute to the advancement of AI algorithms in strategic games.

Overall, this project combines the challenges and complexities of Gomoku with the potential of the MCTS algorithm to create a refined AI solution capable of achieving better performance in the game.

2 Literature Review

The development of AI algorithms for Gomoku has been the subject of extensive research in recent years (Espinoza et al., 2022; Liang et al., 2023). Monte-Carlo Tree Search (MCTS) algorithms have emerged as a prominent approach to address the game's complexity (Chaslot et al., 2008). However, one of the primary challenges lies in the lack of efficient tree expansion strategies, which limits the scalability of these algorithms. This limitation becomes particularly pronounced when dealing with larger Gomoku boards or increasing search depths.

To overcome these challenges, researchers have explored various approaches to refine the MCTS algorithm. Kuhlmann and Stone (2006) introduced heuristics and domain-specific knowledge to guide the tree exploration process, aiming to prioritize promising branches and enhance decision-making capabilities. These modifications have been successful in improving algorithm performance not only in Gomoku but also in other domains such as chess (Helfenstein et al., 2024) and Go (Finnsson and Björnsson, 2011). Inspired by previous studies (Zhao et al., 2012; Tang et al., 2016), valuable insights can be leveraged to develop a refined MCTS algorithm tailored specifically for Gomoku. These studies have explored self-teaching adaptive dynamic programming and employed the MCTS algorithm to enhance Gomoku gameplay.

Furthermore, the field of pattern recognition (Theodoridis and Koutroumbas, 2006) provides a theoretical foundation for understanding the underlying patterns and strategies within Gomoku gameplay. This knowledge can be of great value in improving the decision-making abilities. Moving forward, the implementation of our refined MCTS algorithm will be described in the subsequent section.

3 Implementation

Here, we propose our refined MCTS algorithm.

3.1 Basic MCTS algorithm

Policy in search tree is to determine the next node to start the simulations process. The policy is designed as follows:

- If the current node is not fully expanded, select an unvisited child node as the next state and mark it as visited.
- If the current node is fully expanded, employ the Upper Confidence Bound (UCB) formula (Formula 1) to evaluate and choose the best child node as the next state. Update the current node to this selected child node and repeat the process until the current node becomes a terminal node.

$$UCT = X_j + c\sqrt{\frac{\log n}{n_j}} \qquad (1)$$

where:

- X_j is the average reward of the node.
- c is the exploration parameter.
- n is the total number of simulations for the parent node.
- n_j is the number of simulations for the node.

The **Roll-out policy** uniformly selects the next state from the set of possible legal movements available in the current state. "Legal movements" refer to the empty positions on the current board. The motivation behind this choice is to expedite the simulation process by swiftly exploring a diverse range of potential moves. By uniformly selecting legal movements, the Roll-out policy aims to quickly conduct numerous simulations, maintaining a balance between thorough exploration and computational efficiency. This uniform approach is applied to both the player's and the opponent's Roll-out policies, ensuring a consistent and efficient exploration of the solution space during the simulation phase.

Once roll-out process finished, it will end at the leaf node with the game results. **Back propagation** will carry up the game result to the root node, and for statics of each node on this process path will be updated. Here, statics means X_j (the average reward of the node) in formula 1.

3.2 Refined Roll-out Policy

In the context of the 15x15 Gomoku board game, the computational demands posed by the Monte-Carlo Tree Search (MCTS) algorithm are formidable, with the deepest depth reaching 224. If the node is fully expanded, we apply the following formula to select the next node to start roll out phrase.

Recognizing the impracticality of such extensive computations, we strategically simplify the roll-out policies to make the implementation feasible. In this adapted approach, the algorithm now only explores three levels of depth within the tree, grounded in the information derived from the current explored node. This simplification aims to strike a balance between computational efficiency and algorithmic effectiveness, allowing for a more manageable yet still insightful exploration of potential moves both from competitor and AI view. By pragmatically adjusting the depth of the Monte-Carlo Tree, our research seeks to navigate the challenges posed by computational constraints while maintaining a viable and efficient solution for strategic decision-making in the dynamic environment of the Gomoku game.

3.3 Heuristic Knowledge

As we refine the roll-out policy to address the challenges posed by the Gomoku game, a new concern arises: the evaluation of the current board situation. Fortunately, Gomoku lends itself to a straightforward modeling of rewards. Our proposed model focuses on assessing the proximity to victory in the game. By gauging how close we are to achieving a winning position, we can effectively evaluate the desirability of the current game state. This approach aligns with Gomoku's inherent goal of achieving a sequence of five stones in a row and provides a clear and intuitive metric for assessing the strength of a given board configuration.

Type of Threat	Heuristic Value
Open_1	1
Open_2	10
Open_3	100
Open_4	1e6
Open_5	1e10
Half_Close_1	0.1
Half_Close_2	1
Half_Close_3	10
Half_Close_4	1000
Half_Close_5	1e10
Close_1	0.01
Close_2	0.1
Close_3	1
Close_4	100
Close_5	1e10

Table 1: Reward Points for Different Gomoku Threat Types

Table 1 illustrates the reward points corresponding to various board situations, categorized into three threat classes: Open, half-closed, and closed. In the Open type, there are no opposing pieces within the pointed sequence, while the half-closed type involves one opposing piece in the sequence, and the closed type signifies a sequence entirely blocked by opponent pieces. It is noteworthy that an open 4 threat can lead to victory for both players, whereas a half-open 4 threat can only secure the win for the current player.

We employ a comprehensive counting methodology across vertical, horizontal, and diagonal orientations. This enables us to categorize the threat into three distinct classes based on the outcomes derived from our limited roll-out results.

Now, we modify the Upper Confidence bound in **formula 1** as

$$UCT\left(v_i, v\right) = \frac{H\left(v_i\right)}{N\left(v_i\right)} + c\sqrt{\frac{\log(N(v))}{N\left(v_i\right)}} \quad (2)$$

In our heuristic evaluation, we denote each value as hv_i, representing the heuristic value for a particular board configuration. Additionally, we utilize counting values for both players, denoted as $cv1_i$ and $cv2_j$, where $cv1_i$ corresponds to player one's counting value and $cv2_j$ corresponds to player two's counting value. These counting values are paired with their respective heuristic values.

where

$$H\left(v_i\right) = \sum_i hv_i \cdot cv1_i \; - \lambda \sum_j hv_j \cdot cv2_j \quad (3)$$

The parameter λ serves as the control for the attack coefficient, allowing us to strike a balance between offensive and defensive strategies. By considering the potential moves of the opposing player, we aim to achieve equilibrium between attack and defense, ensuring a well-rounded gameplay approach.

We define $H(v_i)$ as the margin between the points scored by both players on the current board. Our objective is to maximize this margin, as it reflects our advantage over the opponent. This approach underscores our commitment to gaining a strategic edge while considering both offensive and defensive maneuvers.

4 Result

In our test results, we demonstrate the efficacy of our AI by showcasing its ability to make intelligent decisions that effectively balance offensive and defensive tactics.

Through various scenarios and gameplay situations, we highlight instances where our AI strategically prioritizes offensive moves to maximize its chances of winning. Additionally, we illustrate how the AI adeptly switches to defensive maneuvers when necessary, thwarting the opponent's advances and safeguarding its position on the board.

4.1 Test in direct way

In **Figure 1** and **Figure 2**, we illustrate a pivotal moment where the AI decisively makes a "correct"

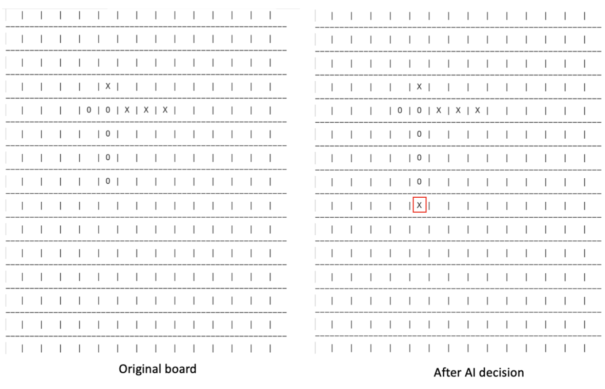

Figure 1: Test for defence

decision. Here, ”correct” for figure 1 signifies not just any advantageous move but specifically the one that is imperative to avert an imminent loss. In **Figure 2**, ”correct” refers to a move directly leading to winning the game. This strategic choice underscores the AI’s exceptional ability to analyze the game state with precision and identify the singular move crucial for its survival.

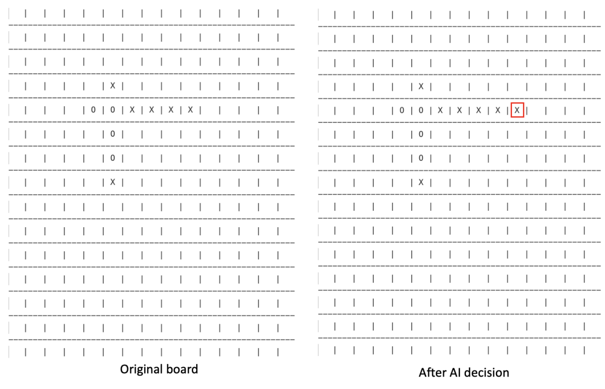

Figure 2: Test for attack

4.2 Test for traps

A vicious opponent may lay a trap for you. In the following context, we will illustrate AI will recognize the traps and make useful countermeasure.

In Figure 3, we observe the opponent player ’O’ executing a strategic maneuver by setting a trap along the horizontal line, and it is a common trick in Gomoku game. This trap poses a significant danger to our AI player, as overlooking it could lead to defeat in just two moves.

In Figure 4, despite the presence of a trap, our AI player can secure victory faster than the opponent. Therefore, our model must carefully weigh the threat posed by the trap against our advantageous position and make a calculated, forward-thinking decision.

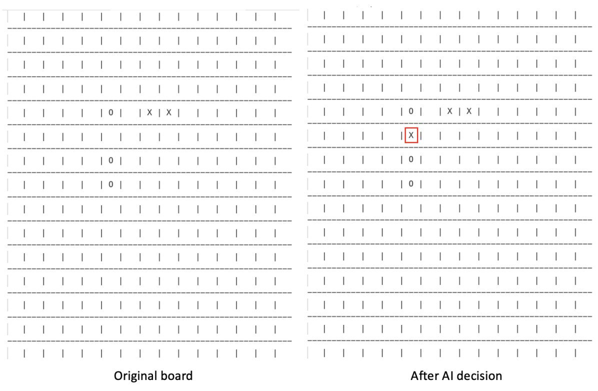

Figure 3: Test for regonize the trap

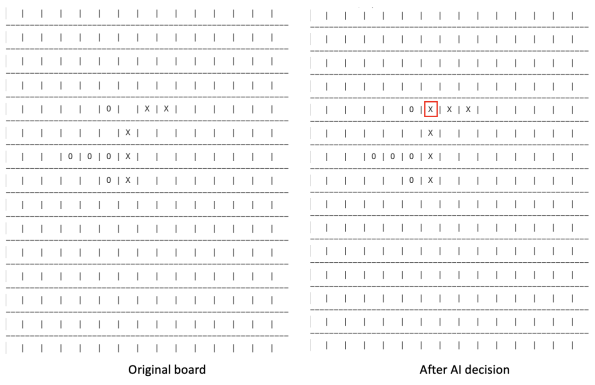

Figure 4: Test balance between defence and attack

Thanks to the predictability offered by our rollout policy, our AI model can anticipate future scenarios several moves ahead. Leveraging this foresight, the AI adeptly identifies the impending threat and makes the necessary corrective moves to neutralize the trap, all while capitalizing on its advantageous position. This decisive action underscores the effectiveness of our AI’s strategic planning and its ability to mitigate risks effectively in the face of cunning opposition tactics.

5 Discussion

In both tests above, we set the total simulation number to 50,000. However, there is a significant discrepancy in simulation time, as shown in Figure 5. Test 2 (Attack test) only took approximately 40 seconds, whereas the other three tests took nearly 150 seconds each. Additionally, Test 2 converged with only 5,000 steps, while Test 1, Test 3 and Test

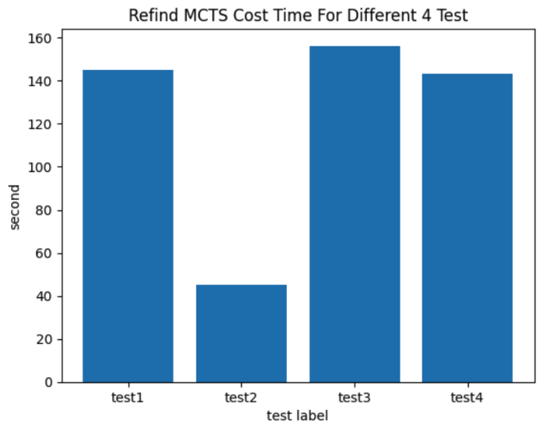

Figure 5: Time needed for 4 tests

4 required at least 30,000 steps to converge. The reason for this difference is that Test 1 are nearly close to winning the game, with only a few tree layers to traverse, resulting in quick convergence. The distinguishable time between Test 1 and Test 2 is because, in Test 2 (Attack test), the AI consistently identifies the winning move in the next step, rendering most rollout and tree search strategies redundant. On the other hand, Test 1 requires defensive moves, leading to a heavily expanded tree and increased computational complexity in future steps.

5.1 Future Work

5.1.1 Adaptive steps

As mentioned above, we found that in some cases our proposed algorithm only need to take a few steps to make intelligent decisions, instead of simulating 50,000 steps. This adaptation could involve dynamically adjusting the number of simulation steps based on the specific characteristics of each test scenario. By incorporating adaptive steps, the algorithm may optimize its performance by allocating computational resources more efficiently, particularly in scenarios where convergence occurs rapidly or where extensive exploration is required.

5.1.2 Bias sampling

One potential enhancement to the search-tree strategy could involve biasing the sampling towards empty places located near the intensive pieces on the board. By prioritizing these areas during simulation and exploration phases, the AI can focus its efforts on positions that are strategically advantageous and more likely to lead to successful outcomes. This targeted approach aims to improve the efficiency of the Monte Carlo Tree Search algorithm by allocating computational resources to areas of the game board that hold greater potential for influencing the outcome.

5.1.3 Deep Learning Method

By amalgamating MCTS with deep learning techniques utilized in AlphaGo, capabilities of our Gomoku AI system may be elevated. Through this integration, deep neural networks can enhance both the evaluation function and rollout policy within the MCTS framework, enabling more informed decision-making and realistic simulations. Moreover, leveraging deep learning for domain-specific knowledge acquisition empowers the AI to discern intricate patterns and strategies, culminating in a sophisticated AI system poised to achieve remarkable performance in the game of Gomoku.

6 Conclusion

In conclusion, our project has ventured into the domain of artificial intelligence and gaming with the ambitious goal of creating an AI system capable of mastering Gomoku, a strategic board game using a 15x15 grid. Our approach revolves around the implementation of MCTS algorithm. Recognizing the inherent challenge posed by the vast computational requirements of fully traversing the Monte Carlo Tree, we have focused on refining our algorithm in two key areas: incorporating distinctive heuristic knowledge tailored to Gomoku strategy and addressing the limitations of roll-out depth. By leveraging game-specific heuristics and optimizing roll-out policies, we have laid a solid foundation for an AI system proficient in navigating the intricacies of Gomoku gameplay.

References

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. 2008. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217.

Jose Luis Vazquez Espinoza, Alexander Liniger, Wilko Schwarting, Daniela Rus, and Luc Van Gool. 2022. Deep interactive motion prediction and planning: Playing games with motion prediction models. In *Learning for Dynamics and Control Conference*, pages 1006–1019. PMLR.

Hilmar Finnsson and Yngvi Björnsson. 2011. Game-tree properties and mcts performance. In *IJCAI*, volume 11, pages 23–30.

Felix Helfenstein, Jannis Blüml, Johannes Czech, and Kristian Kersting. 2024. Checkmating one, by using many: Combining mixture of experts with mcts to improve in chess. *arXiv preprint arXiv:2401.16852*.

Gregory Kuhlmann and Peter Stone. 2006. Automatic heuristic construction in a complete general game player. In *AAAI*, volume 6, pages 1457–62.

Wen Liang, Chao Yu, Brian Whiteaker, Inyoung Huh, Hua Shao, and Youzhi Liang. 2023. Mastering gomoku with alphazero: A study in advanced ai game strategy. *Sage Science Review of Applied Machine Learning*, 6(11):32–43.

Zhentao Tang, Dongbin Zhao, Kun Shao, and LV Le. 2016. Adp with mcts algorithm for gomoku. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE.

Sergios Theodoridis and Konstantinos Koutroumbas. 2006. *Pattern recognition*. Elsevier.

Dongbin Zhao, Zhen Zhang, and Yujie Dai. 2012. Self-teaching adaptive dynamic programming for gomoku. *Neurocomputing*, 78(1):23–29.