Proceedings of the 4th International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/53/20241503

Application of Monte Carlo Tree Search algorithm in Go playing

Wanyi Jiang

Hangzhou International School, No. 2190 Xiangbin Road, Changhe Street, Binjiang District, Hangzhou, China

Jiaw67008@hisdragons.org.cn

Abstract. The reputation of Monte Carlo Tree Search (MCTS) algorithm is built upon its ability to handle a large number of options and make optimal decisions and detailed plans in situations with limited information. It was initially applied in the game of Go, a traditional and complex combinatorial strategy game, which severely challenged the efficiency of traditional brute-force algorithms that relied on simulating every node and branch in the search tree. The success of AlphaGo, developed by Google, represents a significant advancement in artificial intelligence technology, utilizing deep learning networks and MCTS techniques. In this article, we analyze the limitations of traditional brute-force approaches in Go playing and compare them with the application of MCTS, which overcomes these limitations through the iterative process of four steps: selection, expansion, simulation, and back-propagation. The discussion covers the application of MCTS in the game of Go. Besides, we evaluate the advantages and shortcomings of the MCTS algorithm. The article concludes with a summary and prospects for future research.

Keywords: Monte Carlo Tree Search, AlphaGo, Limitations, Application.

1. Introduction

As the oldest broad game that is still incessantly played and well-known worldwide nowadays, Go, also known as weiqi in Chinese, was invented 4,500 years ago in China. The general rule is that the player who has conquered greater territory and gain maximum total scores is considered as winner [1,2,3]. In 11th century, a Chinese scholar calculated the total amount of possible positions in broad of Go in his book Dream Pool Essays, getting a shocking result of 10 to the power of 172, which far surpasses the sum of atoms in universe, which is 10 to the power of 80. Significantly, the simple rule and computational complexity of Go cause the traditional brute-force program infeasible to explore entire complex domain and evaluate sheer number of possible movements in the broad, and that the reason why before first victory of AlphaGo beating a human professional Go player Fan Hui, winner of the European Go Championship, public held a highly skeptical attitude toward the possibility that artificial intelligence could transcend capacity and wisdom of human in this challenging field. Invented by a group named DeepMind Technologies, an organization affiliated with Google, AlphaGo is an outstanding computer program relying upon usage of MCTs Algorithm, which combines the standards of Monte Carlo strategies, applying random sampling and statistical evaluation, and an artificial neural network (machine learning approach) which provides knowledge through massive amounts of trainings, acquiring valuable data from both resources of top human players and computer’s simulative self-play

[4,5]. In March 2016, AlphaGo overwhelmingly beat Lee Sedol by a ratio of 4-to-1, becoming the first program beating 9-dan professional player; At 2017, it won the match competing with the number one ranked player in the world, Kejie [6,7]. A series of incredible successes of this sophisticated program tremendously shocked the world, bringing profound and unprecedented impacts on both communities of chess play and artificial intelligence, forcing humans to alter their previous perspective and abandon out-dated contempt for AI, but considering it in a much more serious and formal manner.

In this article, the limitations of classical brute-force program in Go playing will be analyzed, as its domain-dependence and excessive consumption of computation power on programming non-optimal routes, and on the contrary, how the application of MCTs overcomes these obstacles and achieve significant breakthrough, through repeated iteration of the four steps, including selection, expansion, simulation, and back-propagation, both advantages and shortcomings, and expansive application in area outside of game will also be discussed and evaluated.

2. Limitations of traditional brute-force program in Go playing

2.1. Domain-dependence

Before the significant breakthrough done by AlphaGo, the capacity of traditional artificial intelligence algorithm is restricted and heavily dependent on heuristic knowledge of domain. A standard brute force algorithm is a relatively more direct and straightforward strategy utilized in fields of problem-solving, scheduling, or game playing, immensely relying upon method of exploring all potential possibilities within the domain and eventually providing solution by coming up with the most effective and ideal path [8]. This algorithm built its reputation and popularity based on its convenience and flexibility to be applied to the majority of domains of problems in daily life, and its satisfying effectiveness when handling problems with small area of knowledge [4]. In most computer games, such as chess and tic-tac-toe, the artificial intelligence usually applies graphical representation known as game tree to evaluate each decision of opponent and then determine its own subsequent step, as this game tree is capable of estimating and displaying every possible state of game to predict further progress. Therefore, in most strategy broad games with relatively more manageable number of positions and possibilities, the traditional program with sufficient computing power is capable of beating human players in these games. For example, the brute-force program can easily handle a Western game known as Othello with a broad sized 8 x 8, leading to 10 to the power of 28 estimated legal positions (as a reminder, Go is played in a broad of 19 x 19, resulted in a 10 to the power of 172 legal positions) [6] [8]. On the contrary, as mentioned above, the overwhelming complexity of Go caused by extensive legal positions and possibility contributes to a disappointing computational infeasibility of standard AI methods. For decision-making procedure containing massive amount of knowledge and complex tacks, exploring entire practical domain and traversing the whole tree require enormous time and power of computer system, making the process extremely inefficient and exhaustive during Go playing [4] [6].

2.2. Excessive Consumption of Time and Computing Power

Traditional search programs often encounter difficulties in distinguishing between "bad" or "meaningless" moves, leading to excessive time and computing power being wasted on calculating subsequent possibilities resulting from these unproductive steps. This inefficiency hampers the overall performance of the program. However, a promising solution to this problem lies in the application of MCTs. Unlike traditional search algorithms, MCTs utilize the Upper Confidence Bound formula to strike a balance between exploration and exploitation. This enables MCTs to automatically determine the optimal node and strategy to be selected and expanded [6].

The distinguishing feature of MCTs is their ability to identify and eliminate unsuitable paths efficiently. By discerning between valuable and unpromising moves, MCTs significantly improve efficiency and save valuable computation time. Moreover, this characteristic empowers MCTs to excel in search environments with imperfect or infinite data, making them a valuable tool in various problem-solving domains [6].

2.3. MCTs Algorithm

As a means of overcoming the limitations of traditional game-playing algorithms, various methods have been developed, one of which is the MCTs algorithm. MCTs has demonstrated its effectiveness in games like poker and scrabble, and its integration with deep neural networks has played a vital role in the remarkable achievement of AlphaGo. MCTs relies on random sampling and statistical evaluation, showcasing its remarkable ability to utilize look-ahead tree traversal and sequential decision-making [9,10]. It offers several advantages over traditional AI programs, which heavily rely on knowledge and require complete exploration of the entire decision tree. Unlike these programs, MCTs is remarkably efficient and time-saving. It determines the quality of the next move and makes intelligent decisions without relying on additional information, only relying on the basic rules of the game [6] [8]

This powerful technique extends beyond classic games like Go and can be applied to various modern video games. Its distinguishing feature is its ability to balance exploration and exploitation through the Upper Confidence Bounds (UCB) formula. Extensive simulation iterations are conducted, and the results of these simulations are back-propagated to update the values of each node in the tree. MCTs is mainly consisted of four essential components, including optimal selection of node, expansion of tree by adding chid nodes, simulation as a form of random sampling, and back-propagation of newly updated value to nodes [8].

Step 1: Selection

The MCTs algorithm traverses the tree from the root node by applying a particular strategy. The concept of game tree refers to a diagram illustrating nodes to represent a particular state of game, and edges connecting different nodes indicating the transformation of game states when adding a new move. MCTs uses a formula called the Upper Confidence Bound (UCB) to optimally select nodes which are evaluated as possessing highest estimated value. In this process, MCTs continues to evaluate alternative actions periodically, as an effective way to balance the exploration-exploitation trade-off. This strategy tends to investigate the most promising action sufficiently, ensuring algorithm sticking to the nodes that have the greatest estimated value, in order to discover the best current path. On the meantime, MCTs also does not neglect less promising action with relatively lower winning percentage for the particular consideration that these nodes may be gained less scores due to inaccurate and unlucky simulations [8]. Therefore, it also visits unexplored area. The Upper Confidence Bound (UCB) formula is presented as below,

$$S_i = x_i + C\sqrt{\frac{In(t)}{n_i}} \tag{1}$$

In specific,

$$Si = \ value\ of\ a\ node\ i \tag{2}$$

$$Xi = \ etimate\ value\ of\ the\ node \tag{3}$$

$$C = \ a\ constant\ (bias\ parameter) \tag{4}$$

$$t = \ number\ of\ simulations \tag{5}$$

When the child node with maximum value calculated from UCB formula is selected, it turns to a leaf node, and MCTs enters expansion step [6] [10].

Step 2: Expansion

After we select and reach the optimal node without “child nodes”, we have to create these nodes by expanding current tree. Therefore, a new child node, which has not been investigated in previous iterations, and containing limited available knowledge to indicate its quality and winning percentage, is added to the tree to prepare for simulation [8].

Step 3: Simulation/Rollout

The pseudo-random simulation is an essential step of MCTs algorithm, as in this stage, the algorithm simulates entire game by taking random selections and actions for each player until the simulated game reaches a leaf node, its terminal state, and can take no more actions. At this point, the result is evaluated and generated, as player notices the simulated final state in each side referred to either a win, loss, or draw [8]. In most fundamental version of MCTs, each legal action is selected with exactly even probability, leading to a relatively lower and suboptimal strength in Go playing. Fortunately, in later advanced and updated versions, heuristic knowledge is added to the program, and which enables the moves to become more reliable. As a reminder, the algorithm does not have to create new extra nodes during procedure of rollouts, and thus this process is fast and efficient [6].

Step 4: Back-propagation

After the terminal state is achieved during rollout, the tree will be traversed backward and the estimated value of nodes will be updated, by incrementing its visit count and adjusting reward sum based on result of simulated rollout. We have to update each estimate value backward from leaf node all the way to root node.

Through this search method, MCTS is able to identify high-quality move decisions in the game of Go, with its performance becoming increasingly optimal as the search iterations continue. The successful application of MCTS in Go AI programs such as AlphaGo showcases the potential and efficacy of MCTS in the domain of Go [8].

3. The application of MCTs in the game of Go:

3.1. Move selection

MCTS evaluates different moves by simulating a large number of self-play games to estimate their winning probabilities. It selects moves with higher winning probabilities based on the statistical information. Through iterative simulations and evaluations, the search tree's nodes are continuously updated and adjusted to reflect more accurate winning rate information. Thus, the AI program can make move decisions by leveraging the statistical information provided by MCTS to choose moves with higher winning probabilities [4] [8].

3.2. Position evaluation

The simulation process in MCTS can be used to evaluate winning probabilities for different positions. By repeatedly simulating self-play games and tracking the results, the AI program can gather statistics on wins and loses after each simulation, allowing for the evaluation of the current position. This enables the AI program to assess the advantages and disadvantages of different board positions, facilitating better decision-making [10].

3.3. Introduction of randomness

The random simulation process in MCTS introduces a certain degree of randomness, enabling the AI program to handle uncertainties and complexities present in the game of Go. By incorporating random simulations, the AI program explores a wider range of possibilities, leading to the discovery of better solutions. This inclusion of randomness makes the AI program more exploratory, preventing it from being solely reliant on known optimal solutions and fostering innovative move choices [6].

In summary, the application of MCTs in the game of Go enhances the search and decision-making capabilities of AI programs. By simulating and evaluating a large number of positions, the AI program can select moves with higher winning probabilities and make smarter decisions in complex Go game situations. This forms a fundamental basis and methodology for the development and progress of Go AI.

4. Advantages and disadvantages of MCTs

4.1. Advantages

- **Handling complex and strategic games**

MCTs excels in games with large search spaces, complex dynamics, and strategic decision-making. It relies on statistical sampling instead of exploring complete knowledge of the game state. It is designed as asymmetrical, and the exploration-exploitation trade-off also allows MCTs to optimize exploration and exploitation, to find a balance between discovering new possibilities and exploiting known good actions [8].

- **Updating from simulations**

Through repeated iterations of rollouts, which estimate the value of actions, MCTs is capable of refining knowledge and improving decision-making, allowing it to adapt changing circumstances or evolving strategies, and be sticking to the optimal path with maximum winning possibility [6].

4.2. Disadvantages

The reliability concern is considered as a potential shortcoming of MCTs due to high variance and random nature in each term of rollout [9]. MCTs relies on a massive number of simulations to estimate quality of moves and paths, and therefore an insufficient number of explorations may put algorithm at risk of collecting inaccurate knowledge, missing "good moves", generating inconsistent evaluation, and eventually making non-optimal decisions [4] [6]. MCTs enables alphaGo to win the game when competing with most of chess players, but it is still incapable of defeating world's top players, such as Lee Sedol, without help of CNN, another significant concept in machine learning.

5. Conclusion

This article provides an analysis of the limitations of traditional brute-force search methods in the game of Go and compares them to the application of the MCTS algorithm. MCTS overcomes these limitations through its iterative process of selection, expansion, simulation, and backpropagation. The discussion covers the application of MCTS in the game of Go, including its advantages and disadvantages.

In summary, the application of MCTs in the game of Go has proven to be a significant advancement in artificial intelligence technology. MCTS has addressed the limitations of traditional brute-force approaches by effectively handling a large number of options and making optimal decisions in situations with limited information. The success of AlphaGo, developed by Google, exemplifies the power of combining deep learning networks with MCTS techniques.

Looking ahead, the application of MCTS is not restricted to Go alone. Its potential extends to various domains that involve decision-making in the face of uncertainty and large search spaces. Researchers are exploring the application of MCTS in other complex games and real-world scenarios, such as robotics, automated planning, and optimization.

Despite its successes, MCTS does have some limitations. The algorithm's performance heavily depends on the appropriate balance of exploration and exploitation during the selection process. Fine-tuning this balance and handling certain scenarios, such as long-term strategic planning, remain active areas of research.

Overall, the future prospects of MCTS in Go and beyond are promising. As the field continues to evolve, improvements in computational power, algorithms, and data availability are expected to enhance the capabilities of MCTS and lead to further advancements in artificial intelligence.

References

[1] Bryant, P., Pozzati, G., Zhu, W. et al. Predicting the structure of large protein complexes using AlphaFold and Monte Carlo tree search. Nat Commun 13, 6028 (2022).

[2] Hong, S., Zhuo, H.H., Jin, K. et al. Retrosynthetic planning with experience-guided Monte Carlo tree search. Commun Chem 6, 120 (2023).

[3] Kajita, S., Kinjo, T. & Nishi, T. Autonomous molecular design by Monte-Carlo tree search and rapid evaluations using molecular dynamics simulations. Commun Phys 3, 77 (2020). https://doi.org/10.1038/s42005-020-0338-y

[4] Świechowski, M., Godlewski, K., Sawicki, B. et al. Monte Carlo Tree Search: a review of recent modifications and applications. Artif Intell Rev 56, 2497–2562 (2023). https://doi.org/10.1007/s10462-022-10228-y

[5] M. C. Fu, "AlphaGo and Monte Carlo tree search: The simulation optimization perspective," *2016 Winter Simulation Conference (WSC)*, Washington, DC, USA, 2016, pp. 659-670, doi: 10.1109/WSC.2016.7822130.

[6] J. Kim, B. Kang and H. Cho, "SpecMCTS: Accelerating Monte Carlo Tree Search Using Speculative Tree Traversal," in IEEE Access, vol. 9, pp. 142195-142205, 2021, doi: 10.1109/ACCESS.2021.3120384.

[7] Z. Li, C. Zhu, Y. -L. Gao, Z. -K. Wang and J. Wang, "AlphaGo Policy Network: A DCNN Accelerator on FPGA," in IEEE Access, vol. 8, pp. 203039-203047, 2020, doi: 10.1109/ACCESS.2020.3023739.

[8] Chaslot, G., Bakkes, S., Szita, I., & Spronck, P. (2021). Monte-Carlo Tree Search: A New Framework for Game AI. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 4(1), 216-217. https://doi.org/10.1609/aiide.v4i1.18700

[9] Yang, Z., & Ontañón, S. (2019). Guiding Monte Carlo Tree Search by Scripts in Real-Time Strategy Games. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 15(1), 100-106. https://doi.org/10.1609/aiide.v15i1.5231

[10] Devlin, S., Anspoka, A., Sephton, N., Cowling, P., & Rollason, J. (2021). Combining Gameplay Data with Monte Carlo Tree Search to Emulate Human Play. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 12(1), 16-22. https://doi.org/10.1609/aiide.v12i1.12858