Playing Chess with Monte Carlo Tree Search

Son Tung Do
Dept. of Computer and Information Sciences
Fordham University
New York, USA
sdo3@fordham.edu

***Abstract*—Chess is a popular board game played around the world for centuries. In this work, I will implement a Chess AI using the Monte Carlo Tree Search algorithm. This is a heuristic search algorithm that can guide the tree search in the most promising directions. The Chess program will be tested against multiple AI Chess programs to determine its strength.**

***Index Terms*—chess, board games, monte carlo tree search**

I. Introduction

A. Chess

Chess is one of the most popular board games in history. The game has become a topic of interests for generations of AI researchers, who try to build more and more intelligent machines. Chess is played on an 8x8 board with 6 pieces: pawn, rook, knight, bishop, queen, and king. The goal of the game is to capture the opposing king. A game either ends in a victory for the player who checkmates his opponent's king, or it can end in a draw. A game of chess is played by turns, in discrete time, and has a fully observable state space. The simple rules of chess, in contrast to the complexity of the real world, make it a topic of interest for developing AI algorithms such as searching and learning.

Fig. 1. Chess board starting position

B. Computer Chess

Computer scientists have spent decades studying the game of chess and developing chess computers that can beat humans at this game. They finally achieved a breakthrough in 1997, when IBM Deep Blue defeated the world champion Garry Kasparov [1]. Since then, chess AI has far exceeded the capability of humans. Currently, Stockfish, an expert system AI, is the strongest chess engine in the world [2]. For a brief period of time in 2018, DeepMind AlphaZero managed to beat Stockfish with the use of reinforcement learning [3]. The project was only a proof of concept for developing reinforcement learning algorithms and was not continued further, so currently Stockfish is still the most powerful chess engine in the world.

II. Methodology

A. Heuristic Search

Search is a powerful technique for building an AI system. Exhaustive search of a graph guarantees an optimal solution to any problem we need to solve with that graph. However, when facing a large problem space with limited computing resources, a search algorithm needs to develop heuristic to efficiently search through a graph. Heuristic or best-first search algorithms focus the resource of a computer into searching branches and nodes that are the most promising for that problem. IBM Deep Blue heavily utilized Alpha-Beta pruning, a heuristic search algorithm, in order to narrow the search space the computer needed to perform.

B. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a type of heuristic tree search algorithm developed in 2006 for the purpose of playing Go, a board game with even greater complexity than Chess [4]. It is an iterative search algorithm that gradually builds up its heuristic information through statistical sampling. It achieves this repeating the following 4 steps:

1) **Selection**: The algorithm searches the explored portion of tree. At each level of the search tree, it selects the next node according to a selection policy.
2) **Expansion**: If the selection step does not end in a terminal node, the expansion step adds another node to the explored tree. If the selection step reaches a terminal node, the algorithm skips to the backpropagation step.
3) **Simulation**: The algorithm performs random simulations of the game (known as the Monte Carlo method).
4) **Backpropagation**: The result of the simulation step is propagated back through the tree to update the heuristic.

This process can be repeated as many steps as we want, and number of iterations is a hyperparameter the user can choose, depending on the available computing resource and running time available. The more iterations the algorithm runs, the better and more reliable it will play.

The MCTS selection policy utilizes Upper Confidence Bounds (UCB) to build its heuristic. It searches every possible actions once, then choose the action according to this formula:

$$a^* = \underset{a \in A(s)}{arg\,max} \left\{ Q(s,a) + C\sqrt{\frac{ln\ N(s)}{N(s,a)}} \right\} \tag{1}$$

where

- $A(s)$ is a set of actions available in state s
- $Q(s,a)$ denotes the average result of playing action a in state s in the simulation performed so far
- $N(s)$ is a number of times state s has been visited in previous iterations
- $N(s,a)$ is the number of times action a has been sampled in state s
- Constant C controls the balance between exploration and exploitation

This heuristic balances between exploration ($\sqrt{\frac{ln\ N(s)}{N(s,a)}}$) and exploitation ($Q(s,a)$). Increasing C would make the algorithm value unexplored nodes more than nodes with good values.

III. Experiment and Result

The algorithm is implemented in Python with Pychess as chess simulator. Constant C is set at 1. The program is tested at 100 iterations of the MCTS algorithm (will be referred to as MCTS-100) and played against multiple chess bots on the popular platform Chess.com and Stockfish. At 100 iterations per move, MCTS-100 takes roughly 2 minutes to make a move. The result is presented in Table I. The MCTS-100 program manages to beat two easiest beginner chess bots. With this result, MCTS-100 can be estimated to be at roughly 500 elo.

Opponent	Score
Chess.com AI (250)	2 - 0
Chess.com AI (400)	2 - 0
Chess.com AI (550)	1/2 - 1 1/2
Chess.com AI (700)	0 - 2
Stockfish	0 - 2

TABLE I
MCTS-100 vs other chess bots

This is one example game of MCTS-100 beating the 400-elo Chess.com AI:

1.d4 d5 2.f3 Nf6 3.Nh3 a6 4.c3 b5 5.Bd2 Ra7 6.e4 Ra8 7.exd5 Nh5 8.Bg5 Qxd5 9.Nf4 Qd7 10.Nxh5 h6 11.Nxg7+ Bxg7 12.Nd2 Bb7 13.Be2 Qd8 14.f4 Nd7 15.Rg1 Nb6 16.Qb3 O-O 17.O-O-O Rc8 18.Ne4 Bd5 19.Qb4 Qd7 20.Nc5 Qd6 21.Bg4 Rce8 22.Bd7 Nxd7 23.Nxd7 Qxd7 24.Rd2 c6 25.Re1 f6 26.Bh4 Qa7 27.a4 h5 28.axb5 axb5 29.Qc5 Qa1+ 30.Kc2 Qa4+ 31.Kd3 Bh8 32.c4 Kf7 33.cxd5 Qa6 34.Qxc6 Bg7 35.Qxa6 Bh6 36.Rxe7+ Rxe7 37.Qxb5 Rfe8 38.d6 Ra7 39.g3 Rf8 40.Qc6 Rd8 41.Re2 Bg7 42.Qd5+ Kf8 43.f5 Rc8 44.Re7 Bh6 45.Rxa7 Ke8 46.d7+ Ke7 47.dxc8=Q#

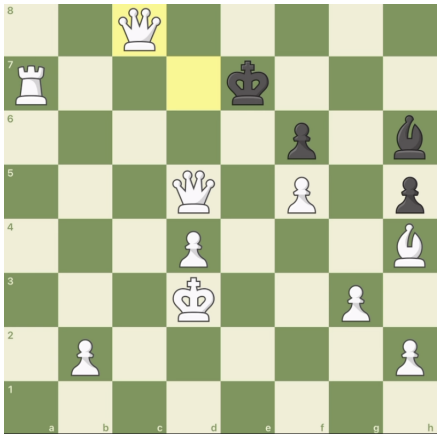

Fig. 2. MCTS-100 checkmate against Chess.com AI (400)

IV. Conclusion

This project demonstrates the power of MCTS for playing the game of chess on its own without any human expertise built in. MCTS is computationally intensive to run due to the amount of repeated simulations needed to evaluate each node, but it is versatile and can be set to run a limited number of simulations only. The algorithm also has an advantage of being highly interpretable, compared to black box neural networks. A human can understand this algorithm by examining the data contained in the explored tree it builds throughout the game.

Further improvement can be made to this chess program either by increased computational power or improved heuristic. The MCTS algorithm can also be used with multiple magnitudes more iterations, which can be achieved on a much stronger computers or parallelized computing across multiple machines. The algorithm can be also be sped up by implementing in high-performance programming languages like C++. The random simulations of MCTS can be enhanced to be more efficient with the use of a neural network and reinforcement learning, which is the approach used by AlphaGo [5]. Such a network can guide the MCTS algorithm to do more efficient simulations but at the cost of immense resources needed for training a combined MCTS + deep reinforcement learning model.

References

[1] M. Campbell, A. Hoane, and F. hsiung Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, no. 1, pp. 57–83, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370201001291

[2] T. Romstad, M. Costalba, J. Kiiski, and G. Linscott, "Stockfish." [Online]. Available: https://stockfishchess.org/

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aar6404

[4] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *Machine Learning: ECML 2006*, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan 2016. [Online]. Available: https://doi.org/10.1038/nature16961