# A Depth-Limited Backtrack Agent based on Markov Probability Inference for Recon Blind Multi-Chess

CS 4649: Robot Intelligence: Planning

Zhangqi Liu, Ruoyang Xu

Last Edited: April 21, 2019

## Executive Summary

**Proposed Method** We have decided to abstract Recon Blind Multi-Chess into two goals and one constraint. Understanding the environment and making the optimal movement in this stochastic environment are the two goals. Time is the constraint for our problem. The proposed method for understanding the environment is probabilistic inference since it has been proven to be a reliable way in perceiving the environment. We plan to use adversarial search, specifically minimax tree search, to achieve the goal. We plan to use Markov assumption and impose search limits to optimize our method for the time constraint. We further propose to accommodate the stochastic environment by using a counter-attack measure that aims to eliminate any enemy piece that captured our piece in the immediate round.

**Details of the Approach** Our agent has two independent sections, sensing and decision making.

The core idea of sensing is probabilistic inference. We modeled the pieces as probability distribution functions that span cross the entire board. At each step, the probability propagates in a Bayesian manner that it only expands on previously visited location, thereby reaching all possible locations. To leave time for decision making, the probability propagation makes the Markov assumption to not consider prior movements. The sensing location is chosen by the place of maximum entropy to maximize information gain. In decision maker, there are two parts: minimax approach and counter attack approach. Counterattack is the greedy attempt to capture any enemy piece that just captured an ally piece. Minimax tree utilizes the possible actions for both ally and enemy pieces to expand the possible chessboard state. The expansion continues until reaching a preset depth or stopped by timer. A scoring function that prioritize capturing enemy high value piece would eventually select the best action.

**Rationale and Result** We employed probabilistic inference and depth-limited minimax tree with backtrack. It is true we lacked the implementation of reinforcement learning, yet we demonstrated our understanding of the course material in the earlier half of the semester by coding up an agent that can defeat the random agent in 86% of the times. Doing this project, we are able to achieve a better understanding of the techniques we employed. We have come to the conclusion that while algorithms may seem simple, the specific representations of the problems that get feed into the algorithms is the essence in engineering solutions.

## Contributions

We divided the project into two parts: sensing and decision making. We agreed to encapsulate our code in a way that it would require minimum interfacing during code integration, which resulted in two individual files: chesspiece.py and chessplay.py, whereas my_agent.py calls the two files. This practice is to ensure code readability in my_agent.py.

Since we both think the other partner's part is more difficult to implement, we consider this project as equally contributed.

**Zhangqi Luo** Responsible for the moving part of the agent. My developments are all in movement.py and chessplay.py. I develop a minimax-based decision maker for the chess AI and a counter attack operation. For minimax, I build a tree generator that extends the board into depth of 3. When extending the board, I simulate the action of both sides and make correct changes in chess

board context. When simulation reaches to the max depth and time limit, I score my context and backpropogate the result for making final decision. The scoring of context is based on the number of pieces left on each side, survival of kings, and how much control my pieces have toward board. For counter attack operation, I program a chess board scanner to find an action that revenge the enemy agent when losing a piece. My decision maker works tightly with Ruoyang's sensing program of enemy pieces positions.

**Ruoyang Xu** Responsible for the sensing part of the agent. Coded the entirety of chesspiece.py, the python instantiation of the sensing model for our agent. Notable contributions include: I wrote an available moves function for all the chess piece at any position used for propagation of probability that any piece is at any location, which is kept as 16 numpy matrices. The chess.Board class is not used due to the immense number of boards I would have instantiate to keep for each possibility (16 pieces $\times$ 8 $\times$ 8 chess.Board objects that each keeps a 8 $\times$ 8 chess board vs a 16 by 8 by 8 matrix). I wrote the move result update for extrapolating if any particular chess piece is killed and/or at any particular position. I also wrote the sensing decision process in that aims to acquire maximize information by targeting the location with max entropy.

## References


[1] Gombolay, M. (2019). Lecture 8 (A Star, MCTS, and AlphaGo) [PowerPoint slides]. Retrieved from https://gatech.instructure.com/courses/39464/files/folder/Lecture%20Slides?preview=3787611

[2] Gombolay, M. (2019). Lecture 9 (Uncertain World I) [PowerPoint slides]. Retrieved from https://gatech.instructure.com/courses/39464/files/folder/Lecture%20Slides?preview=3832791

[3] Gombolay, M. (2019). Lecture 10 (Uncertain World II) [PowerPoint slides]. Retrieved from https://gatech.instructure.com/courses/39464/files/folder/Lecture%20Slides?preview=4110207