Playing Chinese Checkers with IDDFS

Xing Zhao, Bing Han

October 13, 2018

Abstract

We built an AI for Chinese checkers using IDDFS(Iterative deepening depth-first search). The value of each board state is determined via minimaxation of a tree, while the value of each leaf is approximated by weights and features extracted from the board. After optimizing, we have reduces many operation time to deep more steps. The performance of our modified minimax strategy stands out among all the other strategies we have done.

1 Introduction

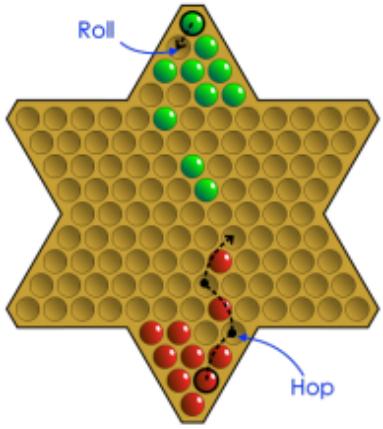

Figure 1: Chinese Checkers board

Chinese checkers is a game played on a hexagram-shaped board that can be played by two to six players individually or as a team. The objective is to be the first to move all fifteen pieces across the board into the opposite starting corners. As shown in Figure 1, the allowed moves include rolling and hopping. Rolling means simply moving one step in any direction to an adjacent empty space. Hopping stands for jumping over an piece into a vacant space(No others in the path). Multiple continuous hops are allowed in one move. A more detailed introduction of the Chinese checkers can be seen in Wikipedia..

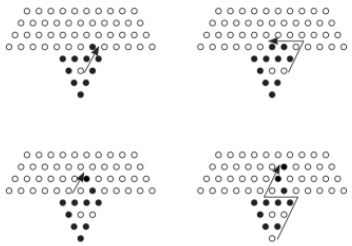

Figure 2: Jump Rules

We tested the performance of different strategies by playing against a random look-ahead Greedy-algorithm agent. We judge the pros and cons of the strategy by the winning percentage and the number of steps required. Moreover, we further modified our strategy such that it divides the game into three stages and applies different strategies thereon. Simulation results showed that ordinary minimax is the optimal strategy.

The rest of this report is organized as the follows. We first talk about how we implemtn the board in Section 2. Then we introduce the basic methodology of our AI in section 3, and point out the difficulties in implementation as well as our solution in Section 4. We cover our modified strategy in Section 5.

2 Board Representation

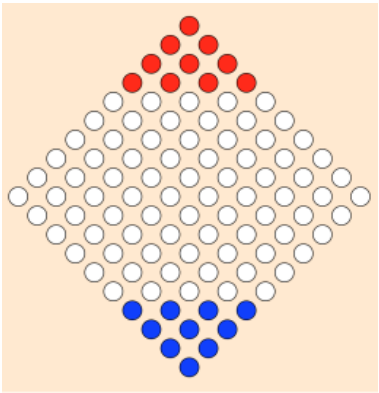

Figure 3: Board Representation

Figure 3 is the starting board where we worked on. Each 0 stands for a vacant spot, 1 stands for a spot occupied by player 1's piece, and 2 stands for a spot occupied by player 2's piece.

3 Methodology

We adopted the classical approach for game playing AI, game search tree, which best mimics the behavior of a human player while demonstrates super-human performance by

taking advantage of the computing power. With each node representing a board state of the game, and each edge representing one possible move from one board state to a subsequent board state, the game search tree can emulate the thinking process of a human player. Chinese checkers is a two-player zero-sum game, thus an objective "value" is needed to evaluate the situation on the board. Player 1's goal is to maximize the value, while Player 2 minimizes it. The logical approach is the minimax tree, which is a decision tree that minimizes the possible loss for a worst case scenario resulted from the opponent's optimal move.

Due to the large state-space complexity of Chinese checker, it it unrealistic to build a top down game search tree. Instead, $\alpha - \beta$pruning is necessary to pruning the tree to reduce much time wasting on the operation. This method return the same result as the Minimax, and it will run as a higher efficency. What's more, we have a time limit–2s, thus our strategy need to make the best result searching in limit time–maximum using the time. So we have chosen the IDDFS(Iterative deepening depth-first search), to refresh the return result in real-time. When finished searching k-depth and there are also some time left, we will continue to search in a deeper tree to consider more steps.

The feature we use to compute value is the distance between pieces and destination.

- A_i, the distances to the destination corner for pieces i of player 1
- B_i, the distances to the destination corner for pieces i of player 2

The evaluation function is the form of $V(s) = \sum_i A_i - \sum_i B_i$ we have also used different strategies in different stages. The detail will be described in the following section.

4 Challenges And Solutions

A Different Strategies

We have tried several strategies to instead the heuristics function. Note that the player1 and player2's pieces seldomly interact with each other. Minimax is a method to consider other's strategy, maybe in the way to stop others. Thus at the beginning and ending of the game, we adopted the maximax strategy to get the fast speed to the destination.

We also have tried different features to measure the evaluation function. Such as the horizontal variance, the vertical variance, the squared sum of the distances and so on. At last, we found the ordinary distance is the best feature we have found. In my opinion, maybe because the hop rule will make the last piece hop to the front and change the evaluation a lot.

B Searching Tree

At the first, we set the constant k-depth tree, and we found that sometimes we cannot return the value because we have a time limit and we don't know sure that how deep we can reach. So we chose the IDDFS to refresh the return value in real-time to make sure that the result is the best since searching.

We also sorted the result as soon as one layer has been searched. And doing this will help to prune tree in the next step. If the time finished but searching not finished, the strategy will return the best result depending on the last layer. If we search the ending of the game, will stop and return the least step way.

When searching depth is more than 1, then using the next evaluation value as the heuristics to sort the action in the next layer. This will accelerate speed of pruning.

C Optimal Operation

In the program running, we have count the time cost every step, and we found that the evaluation function is also waste so many time. So we devote on Optimal Operations to reduce time for search. The strategies we use is a simple one, don't need so many complexity operation. So we think of that we can get the evaluation value quickly using last layer's value. We using hash to store every evaluation value we have compute for each state. Query for hash when we need, if not, then query the last state's value, and compute the difference value to get the current value quickly.

5 Conclusions

In this experiment, we learn and realize a iterative deepening algorithm to fully use the limited time. And in the procedure we are programming, we found we can combine the feature of it with the accumulative evaluation value by using the last depth limit search whose number of layer is 1 less than now. This operation can apparently accelerate the speed of search. We also use similar method like sorting by heuristic before alpha-beta pruning to reduce the searching time.

In the other direction, we considered the feature of the Chinese checker. When the game is about to finish, we observed that the pieces of two players have rare influence with each other which means the piece will only jump through another piece of the same player. So we let the other player's pieces fixed and concentrate agent's attention to search the policy. This operation can apparently improve the agent's ability in the end of the game.

In summary, in the developing procedure, we're moving forward in exploration. There are some critical idea while there are a lot of small optimization. Some of them came from the academic field of minimax search, and the other of them are related to the Chinese checker field. This is a marvellous experience. We are grateful to the teacher and teacher assistant for giving us such an opportunity.

6 CSP Programming

We choose the method of traverse. The code will be given in the zip file. And the idea of programming is written in the py file.

References

[1] Stuart J. Russell, Peter Norvig, *Artificial Intelligence: A Modern Approach Exercise Solutions*