Comp424 AI Report

Lancer Guo

260728557

April 2019

1 Introduction

The aim of this project is to design a AI player for game called Pantago-swap, where each player aims to get a 5 pieces in a row to win the game. AI agent will receive the information of current board, and choose his move based on this information.

This report reflects on how a Pantago-swap AI player was designed and constructed from scratch, demonstrates the algorithms that power the agent. It focus on the discussion about his advantages and disadvantages, and possible improvement that can be done to make the AI behave in a smarter and efficient way.

2 Current Approach And Motivation

2.1 Overview

The strategy that this AI agent uses has two parts. First, within the first 2 turns, it will choose to place the piece to any available center spot, occupy at least 2 center spot to begin with. In the second stage, which will last till the end of the game, all the calculation will be based on Minimax and evaluation function that provided to the agent. It will use Minimax with Alpha-beta pruning to search all the legal moves for as many depths as possible and then return the move with the best value that is calculated based on the evaluation function.

In order to maximize the usage of this approach and make use of our time, a thread is used to control the time limit, inside it uses iterative deepening to conduct a Minimax search with increasing depth until we run out of time.

2.2 Minimax And Alpha-Beta pruning

Minimax is the most straightforward approach that anyone can think of, and also the approach that easily get limited. Minimax provides the ability to see into the future, and produce the most optimal solution while taking opponent's move into consideration.

In this tournament it is safe to assume that other players will also try their best to play optimal. Thus Minimax might be the fastest to find out the best move.

Based on Minimax algorithm learned in class and researched online, Minimax will assure that for each possible state, the selected state will give us the best profit. However, in Pantago-Swap, for each particular move, the branch factor from each move is very large, especially at the beginning of the game. For a single depth there could be more than 180 nodes to check, as a result we might

not even finish the first depth in the worst case.

In order to reduce the time cost, alpha-beta pruning is used to skip all the nodes that will produce the profit worse than the current best profit. At the start of each turn, alpha is initialized to negative infinity and beta is initialized to positive infinity. For each move, alpha is updated if it is on the maximizing node and the new child board state's evaluation is greater than the current alpha. If the updated alpha is greater than the parent's beta, then the rest of the sub-tree can be skipped since beta is the current min value from the parent min node in the search three.

2.3 Evaluation Function

The evaluation function with no doubt is the most important part of this Minimax approach. It is also the key that causes the variety of behaviours even though everyone has a similar implementation of Minimax algorithm. However, sometimes, a very good and detailed evaluation function does not mean a better behaviour will present.

The idea of the evaluation function is fairly straightforward, it uses a number to represents the profit or the win probability of a board state. The rule that it uses to define good and bad is from the rule of this game.

Each time the evaluation function is called, it will first gather information about this board store into a data structure that will be needed for evaluation. The overall score is defined as

$$Overall = PlayerScore - OpponentScore$$

Thus as the second step it will evaluate player's state and opponent's state respectively. If player is the winner in this board state, then the score will be set to the MAX score and return. On the contrary, if player loses then we return the MIN value because it is the worst situation.

For each player, it will evaluate all the pieces that currently presented on board at FOUR direction **Vertical, Horizontal, Left Diagonal** and **Right Diagonal**. For each piece at each direction there are **3** factors that it is looking at:

- Chain Score

  It represents for this piece p, how many other pieces are there so that they can form a chain together. For different length it will have different score. For example when there is 4 in a row, the win probability is much higher than any other, and score is higher.

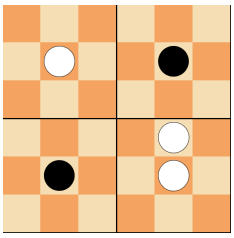
  Other than that, there are two more parameters that it will take account into, *number of space blocked by Boundary*, *number of space blocked by opponent*. For example, a 3 in a row will be very useful if there is 1 side blocked by boundary and the other side empty, because it means we have a 3 in a row in one of the quadrant.

  After evaluation, this score will be set to the score that corresponds to the situation that it satisfied.

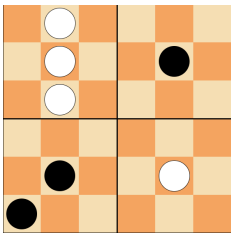
- Center Score

  Since the rule of the game is really similar to Tic-Tac-Toe, it knows the importance role that center pieces are playing in this game. The agent will give extra score for all the piece that can form a chain in the center. As for opponent, it will think that a center piece is more threatening than other piece, so a defensive move should be played to prevent that from happening.

- Swap Score



(a) A 2 in a row swap kill



(b) A 3 in a row swap kill

Swapping is a feature in this game, it makes the game much more complicated, also much more harder to detect the "death trap". As the figure shows, looks like there is no obvious danger, but actually the white piece is just one step away from winning. In figure (a), as long as the White piece form a 2 in a row at top left quadrant and swap TL and TR, then Black is defeated completely. Thus the agent will detect such situation and assign a bigger number so that it can perform the defense accordingly instead of seeing it as a safe state.

The final score for each piece will be calculated as

$$Score = Base * ChainFactor * CenterFactor * SwapFactor$$

Then the score is returned to calculate the overall score of this board.

The evaluation function is also used for Minimax to decide which nodes to expand first at each depth. At the beginning of the depth, it first evaluates all the possible moves, and sort them into a priority queue. For max step it will first process the move that has the highest score, and for min step it will evaluate the one that has the least score. This approach helps the Minimax to prune the sub-tree in a much faster and efficient way.

3 Analysis

3.1 Advantages

- Iterative deepening make the best use of the time and search as many steps ahead as possible
- Play optimally. It is guaranteed to output the best move in the current search.
- Easy to improve. A good configuration of the scoring system will help to improve the performance significantly. It is easy to target on different strategies and opponent type.
- Depending if it is the first player, the strategy and style will be different. For playing white, it uses a evaluation function that cause a more aggressive style. For playing black, it will be more defensive.

3.2 Disadvantages

- The score that assigned to each case is hard to decide. The line between good and bad is really subtle. Even the slightest difference will cause the agent to behave differently, maybe even behave worse.
- It is really slow at the beginning of the game due to the enormous number of possible moves. It might not come up with the most optimal move.

- It assumes that the opponent also plays optimally so it would play really conservatively by picking the best from the worst. If the opponent doesn't it might lose the change to actually play a better move. In this case Minimax is not very useful.

4 Other approach used

Considering the large branch factor that Pantago-swap has, Monte Carlo Search is also a good possible approach. It first select the best node based on the UCT value. Then it expands the node and perform simulation all the way to the end of the game. At last it backtrack and update the win rate of all the parent. The whole procedure is repeated as many times as possible to get a better result. The final move is selected based on the win rate.

During my implementation, from the root, the search is able to visit a lot of nodes at the first depth, but as the depth increases it barely able to simulate many times on children nodes. As the results, the algorithm is not able to select the most optimal move, but reach the sub-optimal move for the most of the time.

5 Improvement

- Using MCTS with Minimax.

  Minimax has limit on seeing all the possible branch, thus we may use MCTS to help us reduce the number of moves that we need to consider. First do MCTS and select the first several best nodes and do Minimax on these nodes.

- We can 30 seconds at the beginning of the first move. We may use MCTS to do a 30 seconds simulation, and use the resulting tree as a reference when we do the Minimax later. Find state that matches the current board state in the tree and do Minimax on the children that has the highest win rate to improve performance.

- Use learning to improve parameter assignments in the evaluation function

- Reinforcement learning