Comp 424
Hus Game AI Project Report

1. Motivation

The goal of this project is to design an AI agent for 2-player game Hus and optimizes the chance of winning by implementing AI algorithms discussed in class.

Several aspects are taken into consideration when I started to form my winning strategy.

Minimax Search

In a 2-player game, the key idea to victory is to maximize the utility of my player and minimize the utility of our opponent. Since the game is perfect information and deterministic, it's easy to simulate all game states for all legal moves, search through each state, and choose the best move we can play. Therefore, minimax search implementation comes to mind.

However, we need to ask ourselves: "How deep into the game tree should minimax search go?"

Timer

Each player has 2 seconds to complete one move (except the first move) and when the time runs out we are forced to take a random move -- this is highly destructive and unfavorable by our side! Thus, some sort of time management must be implemented.

Since we are allowed 30 seconds for our first move, it is also beneficial to set a 30 second timer for our first move to search a deeper depth, especially as a second player.

Naturally, this answers our last question: "We can do minimax search as deep as possible, until our time runs out."

Alpha - Beta Pruning

When considering the time constraint, it is crucial to search as fast as we can in order to get the most value out of the 2 seconds. Thus, alpha-beta pruning is used to skip unnecessary computations and improve computation efficiency.

Evaluation Function

Since the goal of our game is to left our opponent with pits contains 0 or 1 seed only, a simple goal would be to minimize the seeds our opponent got. I.e., to maximize number of our seeds. Thus I choose a simple evaluation function that compute the total number of seeds in my pits.

2. Technical approach

a. When it is my turn to choose move, get current time (start time).

b. Analyze the current board:

   i. Get all legal moves of the board. Simulate each move on current board to get a list of child states.
   ii. Use minimax search with alpha beta pruning to determine which child state is the "best". To do this, I write a recursive function "minimax_ab".

```
function MINIMAX-AB(N, A, B) is
  if N is deep enough then
    return the estimated score of this leaf
  else
    alpha = a; beta = b;
    if N is a Min node then
       For each successor Ni of N
         beta = min{beta, MINIMAX-AB(Ni, alpha, beta)}
         if alpha >= beta then return alpha
       end for
       return beta
    else
       For each successor Ni of N
         alpha = max{alpha, MINIMAX-AB(Ni, alpha, beta)}
         if alpha >= beta then return beta
       end for
       return alpha
    end if
  end if
end function
```

Strong, Glenn. *The Minimax Algorithm*. Rep. N.p., 10 Feb. 2011. Web. 8 Apr. 2016. <https://www.cs.tcd.ie/Glenn.Strong/3d5/minimax-notes.pdf>.

iii. To set up a time constraint, I divided the remaining time by the number of moves I have to look at. For each move I give it a portion of time, and at the beginning of minimax_ab , check if there is enough time. If not then return the best move so far.

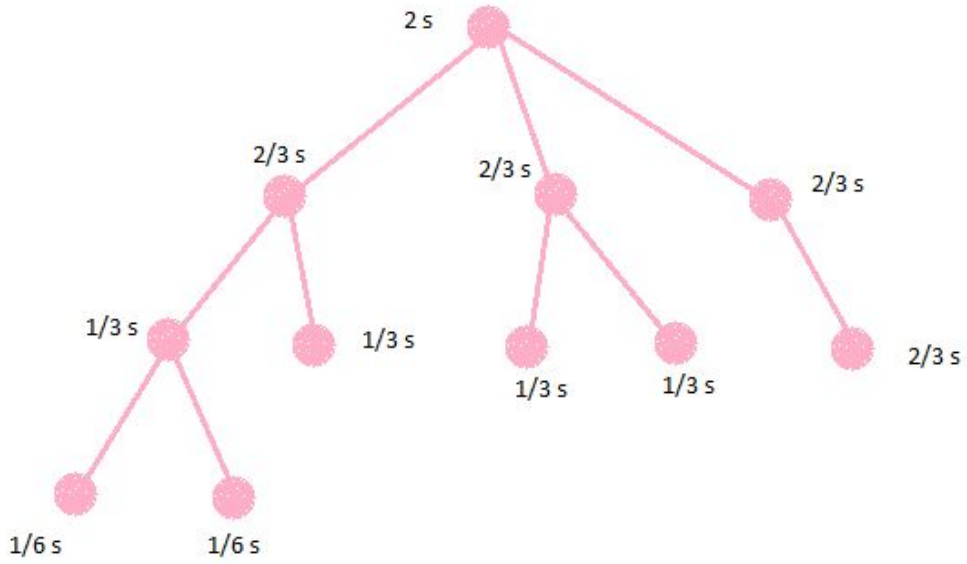

Figure 1. Time management during game tree search

iv. If it is my first move, set the timer to 30 seconds.
v. Evaluate each state by counting the total seeds in my pits.

3. Advantage

a. My agent always choose a move within 2 seconds. Thus no random move punishment.

b. The time allows a dynamic searching strategy. In early game when the move size is relatively large, only 5-6 layer are searched. But when we run to late game or in defensive situation, the number of legal moves are relatively small and we can search deeper to more than 15 levels. That makes our agent very powerful in the late game.

c. With alpha beta pruning we cut off unnecessary subtrees and save time to compute more important nodes. It leads to deeper depth searched per move.

5. Disadvantage

a. My agent do not have strategy to deal with ties.
b. Did not consider the case when my opponent is not optimal.
c. The evaluation function is quite simple which can leads to several best choices.

6. Other approaches I have tried

Different Evaluation Function

Evaluation function: sum of my total seeds and the sum of opponent's pit with only 0 or 1 seeds. This turned out to perform much worse than the final approach. I then tried to add weights to the function.
I.e., evaluation = sum_of_my_seeds * w1 + op_pit_less_than_one * w2.

This approach might have potential after I collect enough data and analyze more matches, but with the many w1 and w2 I chose, the results are not satisfying.

Monte Carlo Algorithm

This approach is to find a way to search deeper in a given period of time.

The agent first search through a low level and find several "possibly best moves" and then perform deeper search on those moves. However, it is very time-consuming to perform minimax search twice. I ended up searching 2 levels deeper on a few moves, but I lost the opportunity to explore the majority of the legal moves, which may also have potential. After experiencing increasing number of loss, I discard this approach.

7. Future improvement

As I addressed earlier in this report, many improvements can be done in the future.

- Find a better evaluation function. This can be done by experimenting with arbitrary parameters, collecting around 1000 matches, and compare the performance of each parameter chosen.
- Use randomization to deal with ties in the game.
- Make the agent "memorize" states with probability of winning, and learn more with each match.

- Improve Monte Carlo technique by modifying the depth and number of potential best moves we want to find in first round of searching, and see what is the optimal number.