Title: Chinese Checkers

Members:

Kaiyang (Kai) Wen — **MarkUs: wenkaiya**

Zejun (Thomas) Yu — **MarkUs: yuzejun**

Roles:

problem encoding: **Thomas**

AI implementation: **Kai**

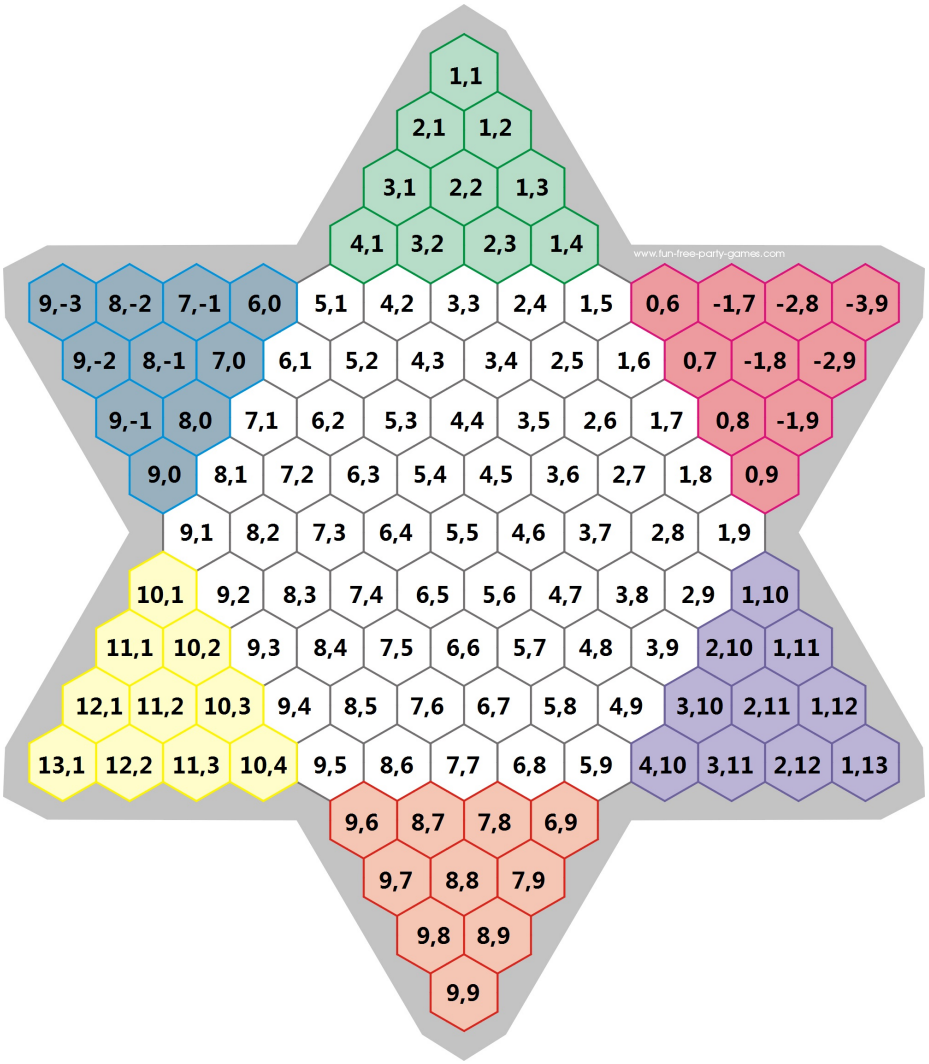
testing: **Thomas**

result analysis: *together*

writeup: *together*

command line interface: **Kai**

Type: Game Tree Search



1 Project Motivation/Background.

This project uses game tree search to play a simplified version of the one on one Chinese Checkers. Full description of this game can be found in the Wikipedia https://en.wikipedia.org/wiki/Chinese_checkers. It is very relaxing to play. This project is suited to game tree search because the one on one Chinese Checkers is a perfect information and deterministic game. Our objective is to write a good enough feature function for the minimax algorithm with alpha-beta pruning.

Note that for simplicity, we eliminated some *nontraditional* Chinese Checkers rules in our project, such as long jumping.

2 Methods.

2.1 Problem Encoding

Unlike other previous approaches which basically hard-code the entire Chinese Checkers chessboard into a 2D array, we designed a tilted coordinate system starting from the corner. Using this coordinate system, we only need to store the twenty stones of each player (i.e. twenty tuples) in order to represent a game state, so it saves more memory as we generate successors of a game search tree. See the picture on the cover page for more detail.

2.2 Artificial Intelligence

Vertical Heuristic

The most common way to modelize the game situation is the vertical distance between the stones to the goal. This heuristic adds up all vertical distances for each side and use the difference as the heuristic. We also modified the traditional distance a little bit: we give less heuristic value when the stone is in its starter area and give more heuristic value when the stone is in the goal area. Doing this will give the pieces more chance to move out of the starter area and move into the goal area. Thus, we are expecting agents have less chance to stuck in the "cheating" situation (see below, we will define cheating later).

The limitation of this heuristic is quite clear. Since we consider the place in the same row as equivalent in this heuristic, it is possible that a piece is moved into some corner place, i.e. (9, -3) and (-3, 9), and thus actually move the stone away from its goal.

Manhattan Heuristic

Manhattan Heuristic is the way we dealt with the previous limitation. Instead of using the summation of the vertical distance, we use the manhattan distance: total number needs to move to the endpoint of the goal area, (1, 1) for black and (9, 9) for white, without jump. The strength of this heuristic is that the agent now considers endpoints of a row being worse than the middle of the row.

Through the testing process, we find a limitation of this method: since the heuristic is hard-coded, sometimes the game will be stuck in some local optimal.

Euclidian Heuristic

Euclidian distance is considered as the squared distance of the Manhhatan heuristic. The idea behind this is as following: not the fastest stone your move into the goal area let you win but the slowest one. By squareing the distance, the difference between rows is decreasing when the stone is approaching the goal. In other words, the agent is more rewarded by moving the "slowest" stone, the one behind of all of other stones.

The limitation of this heuristic is same as the Manhattan heuristic.

Improved Euclidian Heuristic

We are quite stuck at this point. We cannot improve our heuristic now. This Improved Euclidian heuristic is trying to solve the local optimal problem. In addition to the Euclidian heuristic, some randomness is applied here.

We random the power of the Manhattan distance from 1 to 2.125 instead of constant 2 in Euclidian Heuristic. When the power is 1, it is just Manhattan distance. When the power is 2, it is just Euclidian Heuristic. This may give it the ability to jump out the local optimal.

Vertical Heuristic with Splitted Horizontal Displacement[1]

This idea is from Ashish Gupta, a graduate student from Northwestern University. It uses the original vertical distance by adding a horizontal displacement. His idea is that gather stones in the middle of the board will maximize the possibility of "jumping", which apparently move the stones faster towards the goal than move without jumping. Besides, when the game is about to end, setting the stones on the edge of the goal area makes other stones easier to get in the goal area.

The formula of the heuristic is as the following:

Weight Factor * Vertical Heuristic + Splitted Horizontal Heuristic

Also to resolve the local optimization problem, we add some randomness here: we randomrize the weight factor from 3 to 9 every time before we are tring to determine the step we are going to take.

We take the randomness from 3 to 9 because we believe that making progress, here moving closer to the goal area, should be rewarded more than keep stones in the middle.

2.3 Anti-Cheating

By the nature of this game, one player can cheat by leaving at least one stone in one's own corner. In fact, even computer cheats, so we limited the game to be at most 200 turns. We added some simple cheating detection to distinguish the games that don't end in 200 turns.

3 Evaluation and Results.

Since all agents are based on the minimax alpha-beta pruning, then they only differ by $O(1)$, the time for taking products, summations and subtractions. So, it is pointless to measure the performance between each agents, and we should only care about their effectiveness. Also in general, cheating is bad, and so we are hoping our agents not to cheat as often.
We simply use the game results, winning or losing, to represent the effectiveness. We also count total number of games of cheating, and total number of draw games (exceed 200 turns with no agent cheating).

We care about the impact of search depth of the minimax algorithm, and the difference between heuristics.

Black always moves first in all the results.

Agents with different search depth

Strategy	Black	White	B Wins	W Wins	B Cheats	W Cheats	Draw	Total
Vertical	depth 1	depth 3	1	5	3	0	1	10
IM. Euclidian	depth 3	depth 4	0	5	1	0	0	5
IM. Euclidian	depth 4	depth 3	3	2	0	0	0	5

p.s.IM.Euclidian stands for Improved Euclidian

The results above is quite clear that the search depth has positive effect on the effectiveness of a heuristic. The higher the search depth is, the more the agent wins.

Agents with different heuristic

Black	White	B Wins	W Wins	B Cheats	W Cheats	Draw	Total
Vertical	Manhattan	2	6	1	0	1	10
Manhattan	Vertical	10	0	0	0	0	10
Euclidian	Manhattan	9	0	0	1	0	10
Manhattan	Euclidian	1	8	0	0	1	10
IM. Euclidian	Manhattan	10	0	0	0	0	10
Manhattan	IM. Euclidian	0	10	1	0	0	10
Euclidian	IM. Euclidian	3	6	1	0	0	10
IM. Euclidian	Euclidian	4	5	1	0	0	10

p.s.IM.Euclidian stands for Improved Euclidian

As the results suggests, better heuristics show dominant strength when it plays against a worse heuristic. Manhattan Heuristic outplayed Vertical Heuristic. Euclidian Heuristic is better than Manhanttan Heuristic. Improved Euclidian Heuristic is also better than Manhanttan Heuristic. However it seems that we cannot tell which one is better between Euclidian and Improved Euclidian Heuristic.

Versus Vertical Heuristic with Splitted Horizontal Displacement

Black	White	B Wins	W Wins	B Cheats	W Cheats	Draw	Total
	Manhattan	6	3	0	1	0	10
Manhattan		1	8	1	0	0	10
	IM. Euclidian	5	15	0	0	0	20
IM. Euclidian		19	0	0	1	0	20

p.s.IM.Euclidian stands for Improved Euclidian

Here the result suggests that the Vertical Heuristic with Splitted Horizontal Displacement beats the Manhattan Heuristic, but loses to our Improved Euclidian Heuristic. Actually, our Improve Euclidian demonstrated huge advantage when against the Vertical Heuristic with Splitted Horizontal Displacement.

Also, we believe that our randomness helps jump out from the local optimal, because based on our stastics, no draw situation happened when involved an agent with a heuristic with randomness.

4 An Interesting Study

When we look at the stastistics above, it seems that black has somewhat the strength to compensate the disadvantage from the heuristic. Thus, we did some more tests to verify.

Strategy	B Wins	W Wins	B Cheats	W Cheats	Draw	Total
Manhattan depth 2	10	9	1	0	1	20
IM. Euclidian depth 2	10	10	0	0	0	20
Manhattan depth 3	13	6	0	1	1	20
IM. Euclidian depth 3	15	5	0	0	0	20
IM. Euclidian depth 4	1	4	0	0	0	5

The results here differs. It shows that black has advantage when the search depth is 3, game is quite even when the search depth is 2, and maybe has disadvantage when the search depth is 4.

Based on our common sense, we feel that moving first will have some advantage. The result is not consistent with our intuition.

We guess that when the search depth is even, the AI is looking at the heuristic value after the opponent's turn and does the decition based on that heuristic value. But when the search depth is odd, the AI is looking at a turn after his own move. This allows it to play more aggresive, to maximize the output after his own move, and the advantage of moving first applies here.

5 Limitations/Obstacles.

There are two main limitations: the branching factor of this game is too large, and the agents often fall into cycles.

When we were testing, we determined that the branching factor of this game can be more than 112. This restricts our search depth to 3, since when the search depth is 4 the agent runs unreasonably long as the branching factor increases in the mid game, and when the search depth is 5 the agent looks like it is crashed even in the early game.

Also in the early stage of our project, we observed that in some AI vs AI games the agents can move back and forth over the same moves at some point and in some situation of the game. This is due to the identical computed value by the feature function, and in order to overcome we introduced some randomness for each turn an agent moves while not changing the formula of the function. This is sometimes useful for cycle breaking.

There is still a minor limitation. Previously we considered to implement a Monte Caro Tree Search algorithm for this game, but unlike Go the game-tree of Chinese Checkers is actually unbounded. In other words, if two players always move randomly or blindly, then there would be the chance that a game falls into cycles or never ends. We don't have a nice way to sample and randomize the possible moves, and so we decided not to use MCTS. Corollary, we could only impose our knowledge of the game to the agents we built.

Besides, as the time limits, we are not able to get a large scale of statistics. With more testing, the winning rate between agents with different heuristic will be more clear, and we will have a stronger result.

6 Conclusions.

Here are some conclusions we drive from our statistics.

1. The search depth of the minimax algorithm is positively related with the winning rate.
2. Better heuristic can dominant a worse heuristic.
3. When the search depth is 3, moving first gives the agent some advantage.
4. Adding randomization helps jump out of local optimal.

As we are testing the agents, we noticed that many AI vs AI games reach a notably advantageous stage (in the intuitive sense, we don't have a measure for this) in about 65 to 130 turns. This number is slightly larger than human plays the game (according to experience we usually reach a notably advantageous stage in about 40 to 60 turns using only the traditional rule), which we can still build stronger AIs. In all of our agents, we actually hard-coded the base values of the feature functions according to distance or weird insights, and the feature function itself is just a function of these values. (See "stars.py".) When we were developing, "training" the AI is adjusting these base values; neural networks are more popular and perhaps better than humans to do this job, and so we can build a neural network for this game in order to improve.

If we have such a neural network based agent, will it favor cheating (since it is usually considered as a "tie" rather than a "loss") or will it favor winning? This is interesting to bet.

7 Reference

[1] Gupta, A. (n.d.). Retrieved August 11, 2017, from http://www.cs.northwestern.edu/~agupta/_projects/chinese_checkers/web/heuristic_details.htm