Heuristic Analysis

Derek Wang

Jan 1, 2018

Introduction

In this assignment, an implementation of a game playing agent designed to play the game of Isolation was creating to utilize adversarial search techniques. These techniques included the minimax algorithm as the primary decision making algorithm as well as improvements in the form of alpha-beta pruning and iterative deepening to increase the decision making speed and address the horizon problem respectively. States at the maximum search depth per iteration were evaluated using three custom heuristics, each heuristic scoring the states to simulate a different strategy. This report discusses the three custom heuristic functions defined in the Game-Playing Agent Project.

Setup

I ran the tournament program on a computer running Ubuntu and utilizing a CPU with 4 cores and a clock speed of 3.5 GHz. Each run of the tournament process would utilize a significant amount of single core processing power, so I decided to only use 3 cores at one time so as to avoid the impact on performance by other processes while being able to run more tests.

Board Size: 7x7

Move Restrictions: All Chess Knight Moves (L shaped)

Heuristics

Heuristic 1: Chasing Opponent

The first heuristic defined was an example covered in the lecture videos and involved an aggressive approach to playing Isolation. In this strategy, the player would act to prioritize moves that decreased the amount of moves that could be made by the opposing player while maximizing the number of moves the player could make. Using the formula

$$s = \textit{given state}$$

$$h(s) = \#\_player\_moves - (\mu * \#\_opponent\_moves)$$

$$\mu = \textit{weight of aggression (some constant value)}$$

we can value a given state to be higher if the player has a greater number of moves and the opponent has a smaller number of moves. Using a weighted constant μ we can determine how aggressively we want our player to chase the opponent, since either a higher μ or a higher number of opponent moves would more heavily detract from our player's scoring of a given state.

For this heuristic, I tested out different values for μ to see if there was an optimal weighting for the aggressive play style. Testing values of 1, 2, and 3 for μ, I ran the tournament program 100 times,

with custom scoring functions 1 – 3 using a different version of the same heuristic but with different μ values.

μ *Test* Results

μ	Win Rate % Test 1
1	69.9
2	70.97
3	69.4

From this test, I saw that while there wasn't really a significant variation of performance between tests, although $\mu = 2$ stood out as the most consistent best performer. For this reason I decided to use $\mu = 2$ for this heuristic.

Heuristic 2: Prioritizing Own Moves

The second heuristic that I tried involved a similar weight and method as the first but consisted of prioritizing the actions that led to states where we had more options available. Given

$$D(s) = (\mu * \#\_player\_moves) - \#\_opponent\_moves$$

$$\mu = weight\ of\ priority\ (some\ constant\ value)$$

we can expect that instead of chasing the opponent and limiting their moves, we can play to arrive at states where we have the most available moves. The constant μ will drive the player to prioritize options while the subtracting of the opponent moves will allow us to also account for states where the opponent has less moves. This strategy contrasts with the attacking method of heuristic 1 in that it gravitates the player more towards self preservation than causing the opponent to lose.

Heuristic 3: Multi-Strategy (Focus shifting)

This heuristic involves a shifting strategy, where the player acknowledges the end of the game is approaching and shifts the priority of the player to a different strategy. In this case, we could find that it would desirable for the player to first play aggressively following the first heuristic and attempt to limit the amount of moves that the opponent could make. However, as the game progresses the player may want to switch to a different strategy that focuses more on self preservation, much like heuristic 2. Here, we define a potential characteristic heuristics:

$$s = given\ state$$

Aggressive Heuristic

$$A(s) = \#\_player\_moves - (\mu * \#\_opponent\_moves)$$

$$\mu = \textit{weight of aggression (some constant value)}$$

Defensive Heuristic

$$D(s) = (\mu * \#\_player\_moves) - \#\_opponent\_moves$$

$$\mu = \textit{weight of priority (some constant value)}$$

The aggressive heuristic is the same as we have seen in Heuristic 1 while the defensive heuristic focuses more on maximizing the player's own moves as opposed to limiting the opponents. We can combine these two functions by including a weight that changes over the course of the game utilizing the blank space counter.

$$B(s) = \textit{\# of blank spaces for a given state. Strictly decreases}$$

$$M = \textit{Constant \# giving the number of possible spaces on a board (49 for a 7x7 board)}$$

$$h2(s) = (B(s)/M)A(s) + (1-(B(s)/M))D(s)$$

Using this function, as the game progresses, B(s)/M will get smaller as the number of blank spaces decreases. As this happens, the value of 1-B(s)/M will increase proportionally and the player will gradually shift priority from the aggressive strategy to the defensive strategy as the weighting shifts. In either case, the priority of the heuristic will still evaluate to be dominated by a function that is advantageous to it.

Heuristic Performance Comparison

For the cross performance test, I did 3 iterations of the tournament, which defaults to 10 matches against each opponent for each player. I altered the tournament to play 50 matches for each match up to get a more consistent result with each run through. In total, 150 games were played in each match up which results in 1050 games total for each player

Players

Label	Player
AB_Improved	Tournament Default
AB_Custom	Heuristic 1
AB_Custom_2	Heuristic 2
AB_Custom_3	Heuristic 3

Results from 50 Match Tournament

	Player Win Rate %			
Iteration	AB_Improved	AB_ Custom	AB_ Custom_2	AB_ Custom_3
1	71.4	70.0	67.1	72.9
2	61.4	64.3	64.3	67.1
3	74.3	62.9	71.4	81.4
AVERAGE	69.03	65.73	67.6	73.8

Analysis

My initial thought process for picking heuristics was that I needed something that would allow the player to evaluate a state very quickly. Since I wasn't using the extra strategies discussed in lecture for the assignment's game agents, I thought that the best option would be to have a simple heuristic that would take a small amount of time to evaluate, which would allow the player more time to complete more levels in the search. Heuristic 1 was a good inspiration as it provided a decent approach to evaluating a state without having to do more operations, such as looking more layers ahead.

For the best heuristic, I decided that heuristic 3 would be the best option due to the following rationale:

1. Consistently outperformed both Heuristics 1 and 2 as well as the AB_Improved player
2. Does not require any looking ahead, which would have increased the run time and memory usage of the heuristic evaluation. It was important that the heuristic be contained to any given state to avoid extra layers of search.
3. Simplistic design allows for the function to make use of built in game board functions. This also impacts speed as well, since we don't want to do too many computations while evaluating a state. It would be more advantageous if we could quickly determine an evaluation and proceed with deeper layer searching of the state space.
4. The ability for the player to change their game strategy as the state of the game progresses allows the player more flexibility in determining best actions. While there could be improvements as to which heuristic functions should be used within heuristic 3's design, I think it is a strength that the player is more adaptable as the game progresses.