Marifel Barbasa
Udacity AISND March 2018 Cohort
June 2, 2018

Heuristic Analysis on Custom Evaluation Functions for the Game of Isolation

In order to beat the **improved_score** heuristic that outputs a score equal to the difference in the number of moves available between the two players, I decided to start by optimizing the weights of each term in the given **improved_score** heuristic, namely **own_moves** (number of moves available to the current player) and **opp_moves** (number of moves available to the opponent player). In tests 1 through 7, I optimized

h1 = alpha * own_moves - beta * opp_moves,

over the parameters **alpha** and **beta** separately. I determined that **beta** should not be larger than 1. This agrees with the intuition that **own_moves** is more important than **opp_moves**. In summary, when **alpha** is 1, a **beta** value of 0.2 produced the highest win percentage; when **beta** is 1, an **alpha** value of 2 gave the highest win percentage. Overall,

h1_best = 2 * own_moves - opp_moves (Eq. 1),

performed the best and was chosen as **custom_score_2**.

I've also experimented with **own_center_score** (the same **center_score** function in **sample_players.py**) by adding it as another term

h2 = h1_best + own_center_score

but it only decreased the overall win percentage (see tests 2 and 3).

Tests 8 through 14 tested a new idea: extend the base heuristic given in Eq. 1, by looking one step further into the next set of available moves:

h3 = h1_best +

mean_depth_1[gamma * own_move_m - delta * opp_move_m] (Eq. 2),

where **mean_depth_1** calculates the mean over all available moves after making the current move. I expected that optimizing **gamma** and **delta** could only improve the base heuristic. The results concluded that setting **gamma** to 0.5 with a **delta** of 0 produces higher win percentages and outperforms the base heuristic, so that

h3_best = h1_best + mean_depth_1[0.5 * own_move_m] (Eq. 3)

and was chosen as **custom_score**, the best-performing heuristic. Intuitively, smaller values of **gamma** and **delta** were expected because moves further into the future are less important than immediate moves.

I noticed that the **center_score** function in **sample_players.py** is not calculating the center score correctly. The squared distance is given by **(h - y)^2 + (w - x)^2** in the function, and

given that the game board width and height is 7 by default, half the width and height, **w** and **h** respectively, come out to 3.5. However, 3.5 is not the center of the board, given that the coordinates **y** and **x** are 0-indexed. The correct value for **w** and **h** is 3, since position (3, 3) is the exact center of the board on a 7x7 board. Therefore, the correct formula to calculate **w** and **h** should be **(game.width - 1) / 2, (game.height - 1) / 2** on a 0-indexed board. We can further prove that the original formula is incorrect by calculating the squared distance of points (2, 3) and (4, 3) on the board, which should be equivalent. Using the **center_score_improved** formula containing **(game.width - 1) / 2, (game.height - 1) / 2** results in a squared distance of 1 for both points, while the original **center_score** formula containing **game.width / 2, game.height / 2** gives us a squared distance of 2.5 for point (2, 3) and 0.5 for point (4, 3). Therefore, because 2.5 does not equal 0.5, the **center_score** formula is incorrect.

Thus, tests 15 and 16 tested for the best-performing center score functions, and then these center score functions were applied to Eq. 3 in test 17. Unfortunately, the results for the **center_score_improved** function did not exceed the win percentage of **center_score** in any of these tests. On the other hand, the **manhattan_distance_improved** heuristic, which also uses point (3, 3) as the center of the board, appears to perform the best out of the three center score functions and at least as good as the **improved_score** heuristic. The intuition behind the Manhattan (rise plus run) distance is that, since the game of Isolation is played on a grid system and not on the Euclidean plane, pieces have less freedom to move about and in fact, can only move horizontally and vertically, as far as knight pieces go. Therefore, the **manhattan_distance_improved** heuristic

h4 = manhattan_distance_improved

was chosen as **custom_score_3**.

TABLE 1 LEGEND

M --> NUM_MATCHES = number of matches against each opponent
ms --> TIME_LIMIT = number of milliseconds before timeout
Improved Heuristic --> own_moves - opp_moves

TABLE 1

Test #	Bounds	Improved	Custom	Custom	Custom_2	Custom_2	Custom_3	Custom_3
		Win %	Heuristic	Win %	Heuristic	Win %	Heuristic	Win %
1	5M, 150ms	64.3	2 * own_moves - opp_moves	71.4	own_moves - 2 * opp_moves	14.3	own_moves - 3 * opp_moves	14.3
2	5M, 150ms	71.4	2 * own_moves - opp_moves	65.7	own_moves - 0.5 * opp_moves	70.0	own_moves + own_center_score - opp_moves	60.0
3	5M, 150ms	65.7	5 * own_moves - opp_moves	60.0	2 * own_moves + own_center_score - opp_moves	57.1	2 * own_moves + 0.5 * own_center_score - opp_moves	55.7
4	5M, 150ms	61.4	3 * own_moves - opp_moves	65.7	own_moves - 0.8 * opp_moves	72.9	own_moves - 0.2 * opp_moves	77.1
5	5M, 150ms	72.9	1.5 * own_moves - opp_moves	65.7	2 * own_moves - opp_moves	74.3	2.5 * own_moves - opp_moves	70.0
6	5M, 150ms	72.9	2 * own_moves - opp_moves	65.7	2 * own_moves - 0.2 * opp_moves	70.0	2 * own_moves - 0.3 * opp_moves	65.7
7	20M, 150ms	67.5	2 * own_moves - opp_moves	66.8	2 * own_moves - 0.2 * opp_moves	63.6	2 * own_moves - 0.3 * opp_moves	65.4
8	5M, 150ms	64.3	2 * own_moves - opp_moves	71.4	own_moves - opp_moves + sum_depth_1[own_moves_m - opp_moves_m]	52.9	2 * own_moves - opp_moves + sum_depth_1[2 * own_moves_m - opp_moves_m]	68.6
9	5M, 150ms	65.7	2 * own_moves - opp_moves	74.3	2 * own_moves - opp_moves + 0.1 * sum_depth_1[2 * own_moves_m - opp_moves_m]	72.9	2 * own_moves - opp_moves + 0.5 * sum_depth_1 [2 * own_moves_m - opp_moves_m]	64.3
10	5M, 150ms	65.7	2 * own_moves - opp_moves	65.7	2 * own_moves - opp_moves + mean_depth_1[0.5 * own_move_m]	64.3	2 * own_moves - opp_moves + mean_depth_1[0.5 * own_move_m - 0.1 * opp_move_m]	67.1
11	5M, 150ms	71.4	2 * own_moves - opp_moves	71.4	2 * own_moves - opp_moves + mean_depth_1[0.5 * own_move_m]	71.4	2 * own_moves - opp_moves + mean_depth_1[0.5 * own_move_m - 0.1 * opp_move_m]	65.7
12	10M,	65.7	2 * own_moves -	68.6	2 * own_moves -	70.7	2 * own_moves -	66.4

	250ms		opp_moves		opp_moves + mean_depth_1[0.5 * own_move_m]		opp_moves + mean_depth_1[0.5 * own_move_m - 0.1 * opp_move_m]	
13	10M, 250ms	71.4	2 * own_moves - opp_moves	67.1	2 * own_moves - opp_moves + mean_depth_1[1.0 * own_move_m]	61.4	2 * own_moves - opp_moves + mean_depth_1[0.25 * own_move_m]	72.1
14	10M, 250ms	65.7	2 * own_moves - opp_moves	61.4	2 * own_moves - opp_moves + mean_depth_1[0.5 * own_move_m]	70.7	2 * own_moves - opp_moves + mean_depth_1[0.25 * own_move_m]	67.9
15	10M, 250ms	67.9	center_score	66.4	center_score_improved	62.1	manhattan_distance_improved	67.9
16	10M, 300ms	65.0	center_score	67.9	center_score_improved	60.7	manhattan_distance_improved	68.6
17	10M, 250ms	61.4	2 * own_moves - opp_moves + mean_depth_1[0.5 * own_move_m] + 0.1 * center_score	62.9	2 * own_moves - opp_moves + mean_depth_1[0.5 * own_move_m] + 0.1 * center_score_improved	56.4	2 * own_moves - opp_moves + mean_depth_1[0.5 * own_move_m] + 0.1 * manhattan_distance_improved	61.4