Heuristic analysis of a game-playing agent

1. Introduction

An artificial intelligence agent has been developed to play the two-player game of Isolation. In this version of Isolation, each player is restricted to L-shaped moves (like a knight in chess) on a 7-by-7 board. Cells occupied by either player in previous moves are blocked for the remainder of the game, players can jump over blocked positions and the first player with no remaining legal moves loses the game.

2. Methods

Three different custom heuristic evaluation functions had their performance tested against agents provided in the project by running the Python code "tournament.py" on a PC with Intel Core i7-6700K CPU at 4 GHz with 16 GB RAM. For the opponents, the code sample_players.py includes a game-playing agent ("Random") that makes random, legal moves each turn, and a list of other agents that employ either minimax or alpha-beta pruning with various search depths and in-built heuristics ("Null", "Open" and "Improved").

The three custom heuristics are as follows:

Heuristic 1 (herusitic_fun_1) assigns a weighting factor of 4 to the opponent's available moves. It then subtracts the weighted value from the available moves of the current player. The evaluation function is similar to "lecture_heuristic" (in submitted code), which is the heuristic given in lecture. The difference is that it has double the weighting factor. By doing this, the heuristic function essentially penalizes the player more for every remaining move of the opponent.

```
heuristic_value = float(player_moves - 4 * opponent_moves)
```

Heuristic 2 (herusitic_fun_2) divides the number of the player's available moves by that of the opponent. So, the greater the number of remaining moves for the opponent, the smaller the score for the current board state.

```
heuristic_value = float(player_moves/opponent_moves)
```

Heuristic 3 (herusitic_fun_3) divides the number of the player's available moves by the weighted remaining moves of the opponent. This is to see if including a weighting factor in the denominator can increase the performance of the heuristic compared to heuristic 2.

```
heuristic_value = float(player_moves/(2 * opponent_moves))
```

3. Results

The table below summarizes the results from 5 tournaments with 20 matches per opponent. The result was calculated as the percentage of total number of wins over the total number of matches played in each tournament.

Heuristic	Results (%)						
	Run 1	**Run 2**	**Run 3**	**Run 4**	**Run 5**	**Mean**	**STDEV**
ID_improved	69.29	67.14	69.29	67.14	65.00	67.57	1.80
Lecture_heuristic	71.43	63.57	71.43	70.00	68.57	69.00	3.26
Heuristic 1	71.43	70	68.57	75.71	69.29	71.00	2.84
Heuristic 2	70	62.86	69.29	63.57	62.14	65.57	3.76
Heuristic 3	60.71	67.86	67.14	67.14	72.14	67.00	4.09

4. Discussion and Conclusion

Out of the three heuristic evaluation functions tested, **heuristic 1 is the recommended heuristic function for this game**. This is because, based on the existing data, heuristic 1 has a higher mean result % than heuristic 2 and heuristic 3. The lower standard deviation of heuristic 1 also suggests it is more consistent in its performance than the other two custom functions. Furthermore, heuristic 1 has outperformed ID_improved 4 to 1. The results also show that heuristic 1 has a higher mean and lower standard deviation than lecture_heuristic, which suggests that increasing the weight of the opponent's legal moves could improve the evaluation. Nevertheless, it is important to note that heuristic 1 is not the perfect heuristic function, as there remains many more unexplored strategies that could provide better performance than the tested heuristics. The unexplored heuristics could be fundamentally different to the tested heuristics, or they could be minor modifications of existing ones – for example, increasing the weighting factor in heuristics 1 and 3. Overall, it has been demonstrated in this analysis that all custom heuristics have comparable performance with ID_improved.