Heuristic Analysis

The heuristic function chosen is a combination of a ratio based on the number of moves, and the distance from the centre of the board.

(a * count_ratio) + (b * center_distance)

where count_ratio is the ratio between the number of current player moves over the number of opponent moves, and center_distance is the distance of the current player from the center of the board. The a and b parameters in this function determine how much each feature (in this case: count_ratio and center_distance) is able to influence the final value of the heuristic.

Optimising the Heuristic Function

To optimise the parameters (or weights) of the function, the method described as twiddle in the section about Robotics, from the Intro to Artificial Intelligence course on Udacity was used. The optimisation code can be found in the optimize.py file. There are a few modified functions (such as play_round and play_matches) from tournament.py, as well as the twiddle function. The first part of the script finds the parameters with lowest error (highest fitness score). The three best parameters are then used to compete against the baseline agent.

Optimisation Result

The output of running the optimize.py script can be found in optimize.txt. After running the tournament, the win rate for the baseline was 60.0% while the optimised heuristic functions had win rates of 68.6%, 67.1% and 61.4% respectively. The win rates are lower than those reported during the optimisation step, which may be due to the random first move at the beginning of each game.

Final Thoughts

While trying to come up with a good heuristic function, I noticed that even the most primitive functions scored quite well in the tournament. Also, the scores from the tournament weren't very consistent between runs, with the same heuristic functions scoring better and worse against the baseline agent on multiple runs. This leads to the conclusion that the quality of the heuristic function plays a small role in how well the agent performs. The first two moves, which were chosen at random, had high impact on whether the player would win or lose the match. That's why, to considerably improve the performance of the agent, the search algorithm itself should be improved. Caching the computed value of game states and improving the order in which the game states are expanded would make the search look deeper and reach the terminal states more frequently, while playing the first moves according to an opening book would increase the likelihood of a successful match. There are other techniques that could be applied to improve the score of the agent, but ultimately all of them are focused on modifying the search algorithm, rather than coming up with a better heuristic function.