Heuristic Analysis for Isolation Board Game

Executive Summary:

The main objective of this study is to perform a heuristic analysis to find a good evaluation function for the Isolation game-playing agent. Various combinations of probable affecting factors were considered to make the evaluation/heuristic function.

Finally, 'Heuristic_4', an improved version of the given 'improved_score' heuristic was found to perform the best, consistently beating 'AB_Improved' and other opponent agents. The sample size of fair games played w.r.t to each opponent was increased to 30. 'Heuristic_4' performs almost 10% better than 'AB_Improved' agent overall. Also, 'Heuristic_4' beats 'AB_Improved' agent as opponent almost 57% of the times (multiple runs resulted close to this figure of wins).

Introduction:

This report analyses various possible heuristic functions for the board game 'Isolation' which is a deterministic, two-player game of perfect information.

The evaluation function given is:

$$improved\_score = player\_moves - opponent\_moves$$

The objective of this study to find an improved evaluation function which consistently performs better than 'improved_score' defined above.

There were many potential variables I considered for defining various evaluation functions. Some of them are as given below:

1. my_moves – legal moves available to a player at a given state of the game
2. opp_moves – legal moves available to the opponent player at a given state of the game
3. percent_game_completed – a factor to account for the progression of the game
4. dist_from_opp – calculates the euclidean distance of player from the opponent
5. dist_from_center – calculates the distance of player from center of game board.
6. my_max_moves – the maximum no: of moves given a set of legal moves available for the player
7. opp_max_moves – the maximum no: of moves given a set of legal moves available for the opponent
8. my_sum_future_moves – sum of current legal moves + count of legal moves for each of these current legal moves for player
9. opp_sum_future_moves – sum of current legal moves + count of legal moves for each of these current legal moves for opponent
10. my_no_future_moves – count of any losing moves in the list of current legal moves for the player
11. opp_no_future_moves – count of any losing moves in the list of current legal moves for the opponent.

Most of the game loses analysed using the 'sample_players.py' file and the visualization API shows partitioning or walling effect of the board. I found predicting partitioning with simple evaluation functions very difficult. Hence, instead of trying to predict partitioning effect, my attempt has been to change strategy for end game to find the maximum moves to last longest possible.

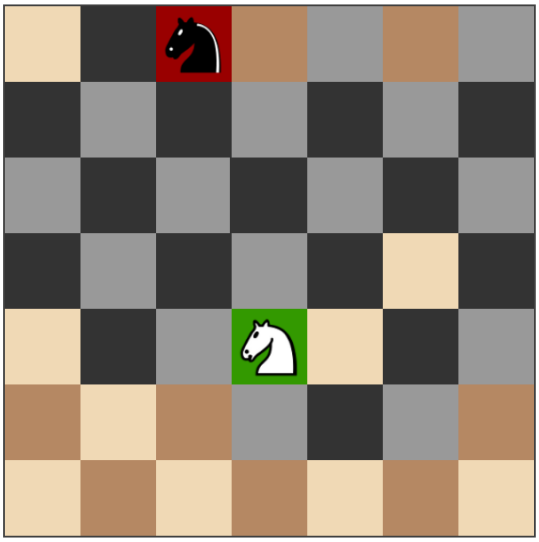


Figure: Example partitioning/walling effect (black knight loses and hence white knight wins)

The Heuristics below detail some of the efforts towards finding a better evaluation function which beats the 'improved_score' function consistently.

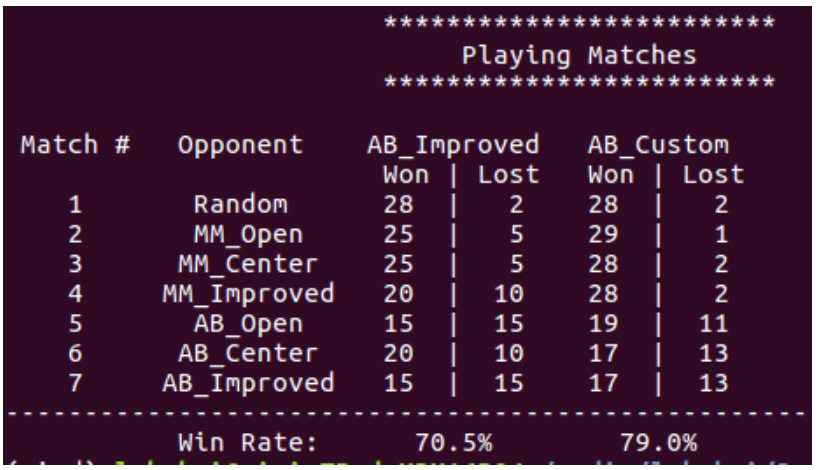
Heuristic_1:

Definition:

$$custom\_score = float(len(my\_moves) - (percent\_game\_complete) * len(opp\_moves))$$

The maximum legal moves when we start the game is 8. As the game progresses, the maximum legal moves reduces to 7 or less. This heuristic is an addition over the given heuristic 'improved_score'. I felt this as a good starting point as 'imporved_score' = len(my_moves) – len(opp_moves) was not really aware of the game progression. At the beginning of the game when there are multiple legal moves available, it really doesn't make a difference when the player chooses to chase after the opponent or not. There are enough moves for both players. Towards the end of the game though, it is better for the player to maximize his own moves and minimize the opponents move, wherever possible.

Performance:



```
**************************
    Playing Matches
**************************

Match #   Opponent    AB_Improved   AB_Custom
                       Won | Lost   Won | Lost
   1       Random       28  |   2    28  |   2
   2       MM_Open      25  |   5    29  |   1
   3      MM_Center     25  |   5    28  |   2
   4     MM_Improved    20  |  10    28  |   2
   5       AB_Open      15  |  15    19  |  11
   6      AB_Center     20  |  10    17  |  13
   7     AB_Improved    15  |  15    17  |  13
-----------------------------------------------
          Win Rate:      70.5%        79.0%
```

This heuristic seems to be working very well as shown above. This function beats 'minimax_only' agents easily. It also does perform adequately well even when the opponent is AB_Improved. Overall, a good improvement.

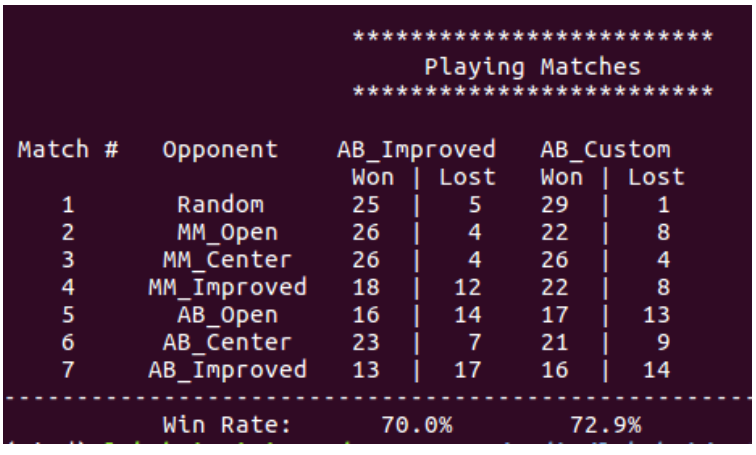
Heuristic_2:

Definition:

$$custom\_score = float((len(my\_moves) + my\_max\_moves - my\_no\_future\_moves) - (percent\_game\_complete) * (len(opp\_moves) + opp\_max\_future\_moves))$$

Trying to improve over heuristic_1, the next option considered was to develop a heuristic based on count of current legal moves and maximum possible future legal moves out of the current legal move. Similarly, the same counts for the opponent is also calculated. My idea was to include a 'move quality' factor into the heuristic so that the player bases its moves not just on the count of legal moves but also on the quality of each of these legal moves. If any current legal move results in a no future move (game loss), that count is subtracted from the above total score. The same is not done for the opponent as we can assume the opponent will most likely not choose that move anyways and hence considering this does not give us much of an advantage (For clarity, I did try both options but this performed better).

Performance:



```
**************************
    Playing Matches
**************************

Match #   Opponent    AB_Improved   AB_Custom
                       Won | Lost   Won | Lost
   1       Random       25  |   5    29  |   1
   2       MM_Open      26  |   4    22  |   8
   3      MM_Center     26  |   4    26  |   4
   4     MM_Improved    18  |  12    22  |   8
   5       AB_Open      16  |  14    17  |  13
   6      AB_Center     23  |   7    21  |   9
   7     AB_Improved    13  |  17    16  |  14
-----------------------------------------------
          Win Rate:      70.0%        72.9%
```

Although this heuristic performs better than 'AB_improved', it does not beat 'heuristic_1'. I believe 'heuristic_2' includes a lot more computations than ideal for a good evaluation function.

Heuristic_3:

Definition:

$$
\begin{aligned}
&if\ percent\_game\_complete\ <\ 0.9:\\
&\quad custom\_score\\
&\qquad =\ float(my\_total\_moves - my\_no\_future\_moves\\
&\qquad -\ percent\_game\_complete * (opp\_total\_moves\ -\ opp\_no\_future\_moves))\\
&else:\\
&\quad custom\_score\ =\ float(my\_max\_moves\ +\ len(my\_moves))
\end{aligned}
$$

From this heuristic onwards, I decided to split the game into two. One strategy for the first 90% of the game and another for the last 10% of the game. The idea was to include an end game tactics. Towards the end of the game when the player would majorly be surrounded by blocked cells, it is essential for the player to maximize its own survival and not consider chasing the opponent. The factor of 90% (%game_completion) was arrived at after various experiments.
For the last 10% of the game, I found that len(my_moves) + my_max_moves works the best, out of the various combinations I tried, for maximizing player benefit and increasing survival. This is probably because it includes the quality of a move using 'my_max_moves' factor together with len(my_moves). For the rest 90% of the game, I changed heuristic 2 to include total current moves and sum of possible future legal moves for each of the current move.

Performance:

```
                         **************************
                              Playing Matches
                         **************************

Match #   Opponent    AB_Improved   AB_Custom
                       Won | Lost   Won | Lost
   1       Random      27  |   3    28  |   2
   2       MM_Open     21  |   9    24  |   6
   3      MM_Center    26  |   4    24  |   6
   4     MM_Improved   20  |  10    24  |   6
   5       AB_Open     16  |  14    20  |  10
   6      AB_Center    16  |  14    16  |  14
   7     AB_Improved   13  |  17    13  |  17
----------------------------------------------------------
          Win Rate:      66.2%        71.0%
```

This seems to be performing better than heuristic_2 but still not as good a heuristic_1 which is much simpler in complexity and better in performance.

Heuristic_4:

Definition:

$$if\ percent\_game\_complete\ <\ 0.9:$$
$$custom\_score = float(len(my\_moves)\ -\ (percent\_game\_complete) * (len(opp\_moves)))$$
$$else:$$
$$custom\_score\ =\ float(my\_max\_moves\ +\ len(my\_moves))$$

The next attempt at improving the heuristic function was to preserve the simplicity of the heuristic_1 and add the end game tactics from heuristic_3.
(Various combinations of variables from heuristic_3 were tried but none performed better than heuristic_1. Hence I decided to go back to improving heuristic_1).

Performance:

```
                      **************************
                           Playing Matches
                      **************************

Match #   Opponent    AB_Improved   AB_Custom
                       Won  | Lost   Won  | Lost
   1       Random       24  |   6    27  |   3
   2       MM_Open      22  |   8    23  |   7
   3      MM_Center     22  |   8    26  |   4
   4     MM_Improved    22  |   8    22  |   8
   5       AB_Open      18  |  12    23  |   7
   6      AB_Center     14  |  16    18  |  12
   7     AB_Improved    14  |  16    17  |  13
----------------------------------------------------------
          Win Rate:      64.8%        74.3%
```

As seen from figure above, the addition of end game tactic did improve heuristic_1 slightly. Overall, this heuristic performs best so far against the given agents including AB_Improved.

Heuristic_5:

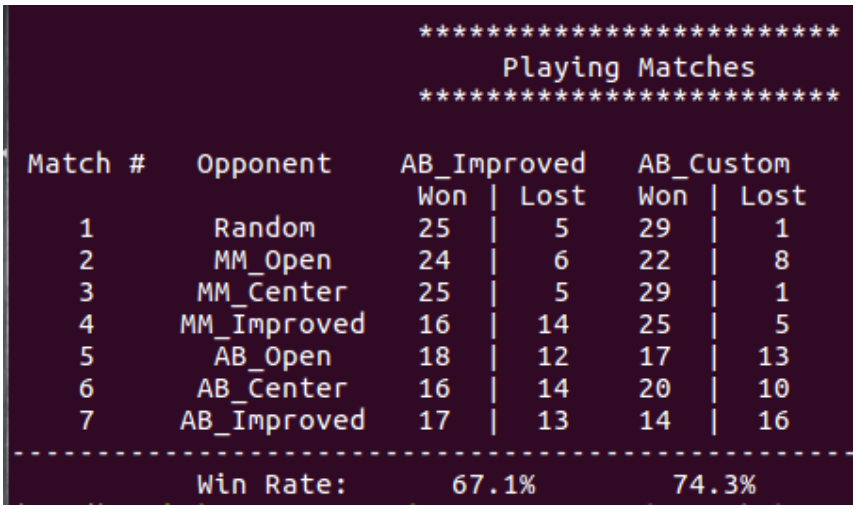
Definition:

$$if\ percent\_game\_complete\ <\ 0.8:$$
$$custom\_score = float(my\_total\_moves\ -\ percent\_game\_complete * (opp\_total\_moves))$$
$$else:$$
$$custom\_score\ =\ float(my\_total\_moves)$$

From heuristic_4, I thought of replacing len(my_moves) and len(opp_moves) with total moves for both players (current count of legal moves and future possible count of legal moves). I also found

that instead of 0.9 as the game completion factor, 0.8 worked better in this case. The end game tactic performed better with my_total_moves instead of my_max_moves + len(my_moves).

Performance:



```
**************************
    Playing Matches
**************************

Match #   Opponent    AB_Improved   AB_Custom
                       Won | Lost   Won | Lost
   1       Random      25  |   5    29  |   1
   2       MM_Open     24  |   6    22  |   8
   3      MM_Center    25  |   5    29  |   1
   4     MM_Improved   16  |  14    25  |   5
   5       AB_Open     18  |  12    17  |  13
   6      AB_Center    16  |  14    20  |  10
   7     AB_Improved   17  |  13    14  |  16
-----------------------------------------------
          Win Rate:      67.1%        74.3%
```

The heuristic is simple and performs well too. But it still doesn't beat the simpler heuristic_1 or heuristic_4 (improved version of heuristic_1). Hence, no further trial to improve this was done.

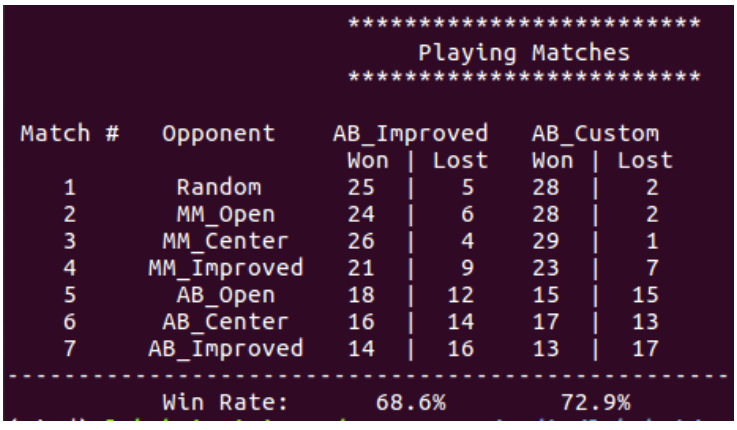
Heuristic_6:

Definition:

$if\ percent\_game\_complete\ <\ 0.9$:
    $\#\ distance\ to\ center\ factor\ added$
    $custom\_score$
        $= float(len(my\_moves)\ +\ percent\_game\_complete * center\_dist\_factor$
        $-\ (percent\_game\_complete) * (len(opp\_moves)))$
$else$:
    $custom\_score\ =\ float(my\_max\_moves\ +\ len(my\_moves))$

This was another attempt to improve heuristic_4. I added a factor to prefer moves further away from the centre at the beginning of the game so that the more moves are left near the centre towards the end of the game. This might avoid cornering or island partitioning towards the end of the game. For the last 10% of the game, the player just tries to maximize his moves based on the maximum future moves available like the previous heuristics.

Performance:



```
************************
    Playing Matches
************************

Match #   Opponent    AB_Improved   AB_Custom
                       Won | Lost   Won | Lost
   1       Random      25  |   5    28  |   2
   2       MM_Open     24  |   6    28  |   2
   3      MM_Center    26  |   4    29  |   1
   4     MM_Improved   21  |   9    23  |   7
   5       AB_Open     18  |  12    15  |  15
   6      AB_Center    16  |  14    17  |  13
   7     AB_Improved   14  |  16    13  |  17
-----------------------------------------------
          Win Rate:      68.6%        72.9%
```

This heuristic also does perform better overall but doesn't beat the simplicity of heuristic_1 or improved heuristic_4. I believe this is due to the fact that adding too many variables makes it computationally complex and hence loses out on performance.

Conclusion:

Multiple combinations of probable variables listed at the beginning of this report were tried. Most of the non-performing ones were excluded from this report. Top six performing heuristics were selected to be presented for this study.

The heuristics are ordered as follows in the game_agent.py file based on their descending order of performance.

1. custom_score() - Heuristic_4
2. custom_score_2() - Heuristic_1
3. custom_score_3() - Heuristic_5
4. custom_score_4() - Heuristic_3
5. custom_score_5() - Heuristic_6
6. custom_score_6() - Heuristic_2

Performance Table:

AB_Improved (% wins)	Custom Heuristic	Custom_score (% wins)	Difference
64.8 %	Heuristic_ 4	74.3 %	9.5 %
70.5 %	Heuristic_ 1	79.0 %	8.1 %
67.1 %	Heuristic_ 5	74.3 %	7.2 %
66.2 %	Heuristic_ 3	71.0 %	4.8 %
68.6 %	Heuristic_ 6	72.9 %	4.3 %
70.0 %	Heuristic_ 2	72.9 %	2.9 %

Heuristic_1 and Heuristic_4 performs the best as shown below.

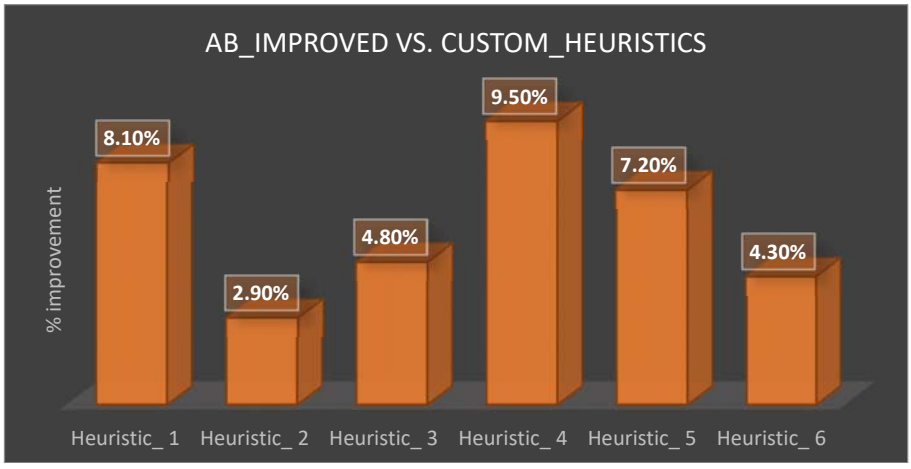


Figure: Heuristic Performance % difference w.r.t 'improved_score'

Although heuristic_1 is the simplest, I would chose its improved version heuristic_4 even though it adds only a bit more performance. This is due to the following:

1. Heuristic_4 considers an end game strategy which I believe is very essential for success in board games. Instead of removing this, I would like to try to improve this strategy further as future work.
2. Heuristic_4 although a bit more complex than heuristic_1 is still simple enough to calculate compared to the various other combinations tried.
3. Heuristic_4 performs the best, consistently beating not just minimax and other alpha-beta (AB) agents but also the 'AB_Improved' opponent.
4. Heuristic_4 maximizes player options by choosing max_moves along with len(my_moves) towards the end of the game. This adds a 'quality' factor to the move and helps the player to go after moves with maximum future moves, hence, maximizing its options for survival.
5. In code, heuristic_4 also considers the option of avoiding one length moves which although might have the maximum future moves but is in the opponent's move_list. Otherwise, the opponent might move into this position essentially ending the game with a loss for the player.

Improving the heuristics further but at the same time keeping it computationally simple is a challenging task. Adopting different strategies for various game progression states by optimizing various combinations of the heuristics might be a good future work.