Heuristic Analysis for Isolation Game

By Uttam Panda

Testing the different heuristics was hard and time-consuming. As the starting positions in the tournament are random, it is hard to get reproducible results, even when increasing the number of matches to 100.

Data available to calculate the heuristics
For the heuristics to be fast enough to compute, I've ended up using information that was easily available:
- Number of available moves to each player
- Position of each player on the board, especially their distance to the center
- Number of blank spaces on the board: can be used to know if we are closer to the beginning or the end of the game. The strategy can depend on how far we are in the game.

I've also tried to calculate the longest available path available to each player when there are few blank spaces left in the game. However, the heuristics was too slow, timing-out and not performing well at all, so I discarded it.

Heuristics 1 (AB_Custom)

Formula:

```
blank_spaces = len(game.get_blank_spaces())
total_spaces = game.width * game.height
return float((blank_spaces/total_spaces)*100)
```

Result: 60 % v.s. 69.6 % (- 9.6%)

Heuristics 1 (AB_Custom) is the worse of the heuristic of all the heuritsics

Heuristics 2 (AB_Custom2)

Formula:

```
my_moves = len(game.get_legal_moves(player))
opponent_moves = len(game.get_legal_moves(game.get_opponent(player)))
corners_array = [(0,0), (0, game.height - 1), (game.width - 1, 0), (game.width - 1, game.height - 1)]
if game.get_player_location(player) in corners_array:
    # downgrade, because of bad position on the gameboard
    my_moves -= 3
return float(my_moves - opponent_moves)
```

Result: 70.1% v.s. 69.6 % (+0.5%)

Heuristics 3 (AB_Custom3)

Formula:
my_moves = len(game.get_legal_moves(player)) opponent_moves = len(game.get_legal_moves(game.get_opponent(player))) return float(my_moves - 2 * opponent_moves)

Result: 71.3 % v.s. 69.6 % (+ 1.7%)

Heuristics recommendation for Heuristics 3 (AB_Custom3):

My recommended heuristics is **Heuristics 3 (AB_Custom3)** combination of number of available moves and players' positions on the board.

Formula:
my_moves = len(game.get_legal_moves(player)) opponent_moves = len(game.get_legal_moves(game.get_opponent(player))) return float(my_moves - 2 * opponent_moves)

Reasons for selecting Heuristics 3 (AB_Custom3):
- Best score: even though it was close to other heuristics, **Heuristics 3 (AB_Custom3)** performed the best in the tournament.
- Short execution time:
    - It uses data that is accessible quickly
    - It uses fast operations
- **Heuristics 3 (AB_Custom3)** does not timeout and enables the search to go deep in the tree, leading to better results.
- **Heuristics 3 (AB_Custom3)** uses information about player and opponent: in an adversarial game, it makes sense to use information about us but also about the opponent.

Summary of the results of all heuristics:

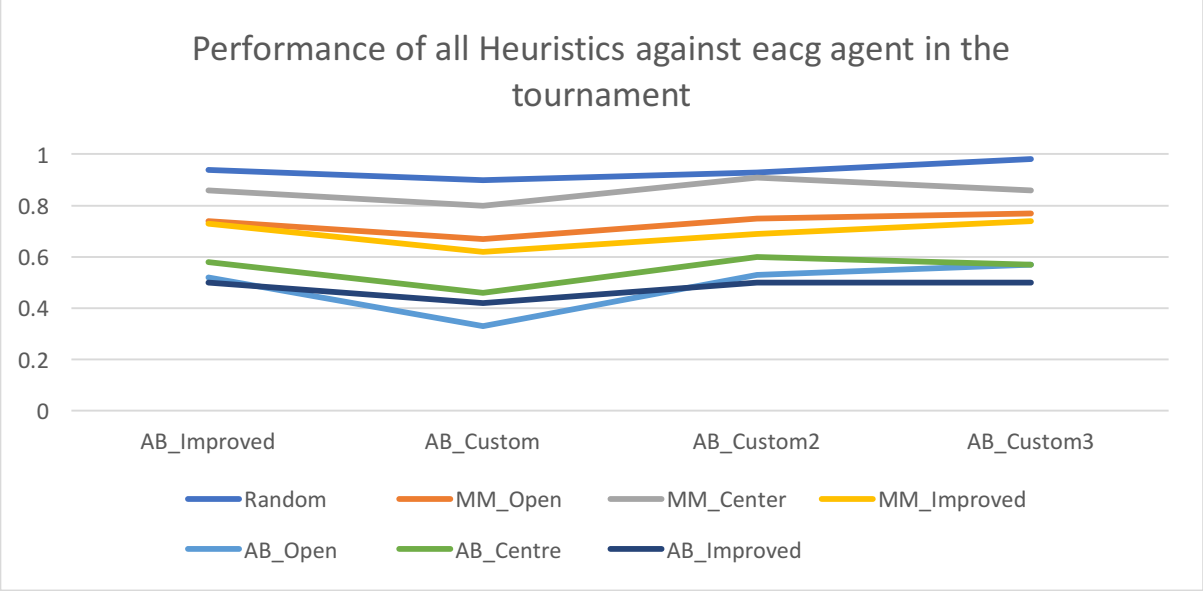
Match	Opponent	AB_Improved			AB_Custom			AB_Custom2			AB_Custom3		
		Won	Lost	Rate %	Won	Lost	Rate %	Won	Lost	Rate %	Won	Lost	Rate %
1	Random	94	6	94	90	10	90	93	7	93	98	2	98
2	MM_Open	74	26	74	67	33	67	75	25	75	77	23	77
3	MM_Center	86	14	86	80	20	80	91	9	91	86	14	86
4	MM_Improved	73	27	73	62	38	62	69	31	69	74	26	74
5	AB_Open	52	48	52	33	67	33	53	47	53	57	43	57
6	AB_Centre	58	42	58	46	54	46	60	40	60	57	43	57
7	AB_Improved	50	50	50	42	58	42	50	50	50	50	50	50

Win Rate %	69.6	60.0	70.1	71.3

Match	Opponent	AB_Improved	AB_Custom	AB_Custom2	AB_Custom3
		Winning Rate %	Winning Rate %	Winning Rate %	Winning Rate %
1	Random	94	90	93	98
2	MM_Open	74	67	75	77
3	MM_Center	86	80	91	86
4	MM_Improved	73	62	69	74
5	AB_Open	52	33	53	57
6	AB_Centre	58	46	60	57
7	AB_Improved	50	42	50	50
Win Rate %		69.6	60.0	70.1	71.3

Analysis of the tournament agents:

The rank of the opponents in the tournament is always the same, no matter which heuristics is used:
The y-axis represents the % of wins of the heuristics so the worst agents will have high values.
Consistently we observe that:



Notes : I'm running on a MacBook Pro, 2 GHz Intel Core i5