Self Training

September 10, 2017

Isolation Heuristic Analysis

Ngoc Minh VU

Version 0.1

– Draft –

Abstract

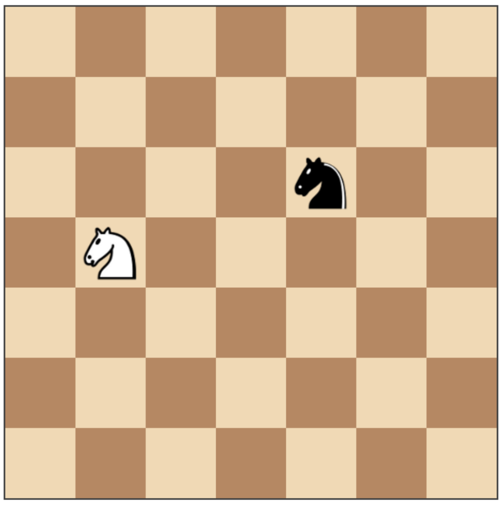
We analyse the performance of different heuristic evaluations using `tournament.py` and summarize their results in this report.

Contents

1 Problem Context - Isolation

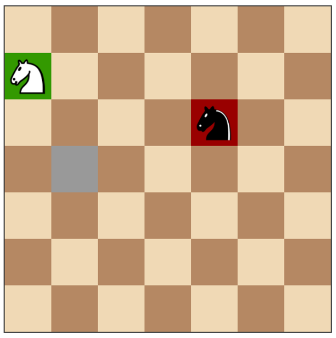
In this project, we want to build a Game-Playing Agent to play Isolation game. However we uses a modified version of Isolation where each agent is restricted to L-shaped movements (like a knight in chess) on a rectangular grid (like a chess or checkerboard). Here is an example of an opening move

Figure 1: Modified Isolation Opening Move: white (3,1) black (2,4)

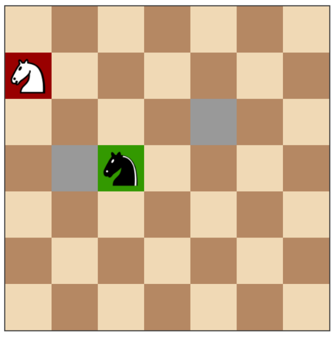


The agents can move to any open cell on the board that is 2-rows and 1-column or 2-columns and 1-row away from their current position on the board. Movements are blocked at the edges of the board (the board does not wrap around), however, the player can "jump" blocked or occupied spaces (just like a knight in chess). Here is an example for few next moves

Figure 2: Next Few Moves



(a) Next move 1: white (3,1) -> (1, 0)



(b) Next move 2: black (2,4) -> (3,2)

2 Heuristic Evaluation

2.1 Aggressive player

As suggested in lecture, one can try aggressive player i.e penalize opponent weight more e.g

$$\texttt{eval}_1(game, player) = \texttt{\#player-move} - w \times \texttt{\#opponent-move}$$

with a weight $w > 0$.

This is implemented in `game_agent.weighted_score` and used in `game_agent.custom_score_2` with a default $w = 1.5$.

2.2 Weighted move

In above evaluation, for each move in the next legal moves we give the same weight of 1, one could think that each move might have different weight. One simple way to assign weight to a move is to use the number of next available moves from it

$$\texttt{eval}_2(game, player) = \sum_{m \in \texttt{player-move}} \#\texttt{move}(m) - w \times \sum_{m \in \texttt{opponent-move}} \#\texttt{move}(m)$$

where $\#\texttt{move}(m)$ is number of move from m and w is a positive weight.

This is implemented in `game_agent.weighted_move_score` and used in `game_agent.custom_score_3`.

2.3 Combining strategy

We can combine the two above score with a mixing parameter λ

$$\texttt{eval}_3(game, player) = \texttt{eval}_1(game, player) + \lambda \times \texttt{eval}_2(game, player)$$

This is implemented in `game_agent.custom_score_1`.

3 Tournament results

Now we will try each heuristic evaluation with different parameter and summary obtained results.

Note that, we increase number of game from 5 to 50 since we find that 5 games is too small for measure the player's performance. Each heuristic is measured against `AB_improved` and they all play again the following cpu agents

- Random: random select a move
- MM_Open: minimax with heuristic evaluation number of open moves
- MM_Center: minimax with heuristic evaluation distant to center
- MM_Improved: minimax with heuristic evaluation difference of player's open moves v.s opponent's open moves
- AB_Open, AB_Center, AB_Improve: alphabeta with corresponding heuristic evaluation

These heuristic evaluations are implemented in `sample_players.py`.

3.1 Aggressive player: $w = 0.5,\ 1.5,\ 2.0$

We obtain the following result

Match	Opponent	AB_Improved		AB_Aggr_0.5		AB_Aggr_1.5		AB_Aggr_2.0	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	88	12	82	18	81	19	86	14
2	MM_Open	68	32	61	39	65	35	69	31
3	MM_Center	75	25	74	26	79	21	80	20
4	MM_Improved	59	41	68	32	70	30	65	35
5	AB_Open	48	52	52	48	51	49	53	47
6	AB_Center	60	40	61	49	60	40	59	41
7	AB_Improved	47	53	61	49	60	40	59	41
	Win Rate:	63.6%		64.6%		64.9%		64.4%	

Look at above result, we observe that $w = 1.5$ gives optimal win-rate (but not very decisively).

3.2 Weighted move: $w = 0.5,\ 1.0,\ 1.5,\ 2.0$

We obtain the following result

Match	Opponent	AB_Improved		AB_Wm_0.5		AB_Wm_1.0		AB_Wm_1.5		AB_Wm_2.0	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	83	17	84	16	84	16	79	21	80	20
2	MM_Open	67	33	71	29	80	20	77	23	67	33
3	MM_Center	73	27	73	27	84	16	87	13	83	17
4	MM_Improved	62	38	67	33	73	27	72	28	62	38
5	AB_Open	49	51	51	49	52	48	51	49	61	39
6	AB_Center	62	38	60	40	57	43	60	40	63	27
7	AB_Improved	46	54	64	36	53	47	48	52	49	51
	Win Rate:	63.1%		67.1%		69.0%		67.7%		66.4%	

Look at above result it's clear that weighted-move evaluation performs better than `AB_Improved` and the optimal weight is 1.0. This makes sense since using the next available moves we can measure each move differently which is sensible.

3.3 Combine strategy: $\lambda = 0.5,\ 1.0,\ 2.0$ and adaptive

Now we test the combine strategy with $\lambda = 0.5,\ 1.0,\ 2.0$

We obtain the following result

Match	Opponent	AB_Improved		AB_Comb_0.5		AB_Comb_1.0		AB_Comb_2.0	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	78	22	85	15	82	18	88	12
2	MM_Open	64	36	67	33	72	28	71	29
3	MM_Center	75	25	85	15	79	21	79	21
4	MM_Improved	65	35	59	41	69	31	77	23
5	AB_Open	52	48	56	44	56	44	55	45
6	AB_Center	58	42	56	44	56	44	55	45
7	AB_Improved	50	50	53	47	49	51	57	43
	Win Rate:	63.1%		66.4%		66.6%		70.4%	

Then later we try adaptive versions

$$\texttt{eval}_{occ}(game, player) = occ \times \texttt{eval}_1(game, player) + (1 - occ) \times \texttt{eval}_2(game, player)$$
$$\texttt{eval}_{1-occ}(game, player) = (1 - occ) \times \texttt{eval}_1(game, player) + occ \times \times \texttt{eval}_2(game, player)$$

where occ is percent of occupied cell in the board.

This evaluation is implemented in `game_agent.custom_score_adaptive`. We obtain the following result

Match	Opponent	AB_Improved		AB_Comb_Occ		AB_Comb_1-Occ	
		Won	Lost	Won	Lost	Won	Lost
1	Random	82	18	87	3	85	15
2	MM_Open	63	37	71	29	76	24
3	MM_Center	73	27	75	25	76	24
4	MM_Improved	58	42	70	30	68	32
5	AB_Open	55	45	55	45	57	43
6	AB_Center	53	47	60	40	61	39
7	AB_Improved	52	48	52	48	51	49
	Win Rate:	62.3%		67.1%		69.3%	

3.4 Chosen evaluation score

After looking at statistical result we will use combination strategy with $\lambda = 2$ as final evaluation heuristic since

- it was based on the improved weight and we assign a reasonable weight for each next legal-move
- the evaluation is straightforward so it will run fast
- it performs best against other player and scores a win-rate of 70.4%.

This is implemented in `game_agent.custom_score`.

4 Conclusion

Examine the above heuristic we can see when we measure a move better, we obtained a better winning rate. However our heuristic evaluation is still very simple hand-crafted function. In the future we might need to investigate how to use reinforcement learning to make it to learn and improve via self-play.