# Heuristics analysis for isolation game

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

I came up with a few heuristics that beat the ID_Improved heuristics consistently.
On average, my final heuristics all explore between 8 and 9 levels deep in the tree.

I ran tournaments on 100 matches and, in case I ran tournaments several times, I averaged the results. For instance, results of ID_improved went from 74.21% to 75.25% with an average of 74.73%.
The best performing is **H4**, winning 77.70% of the games (+2.96% compared to ID_Improved)

#### Heuristics 1 (H1): Linear combination based on blanks (aggressive, then defensive)

Implemented in linear_combination_moves_score.

Formula:
(max_blanks-blanks)*own_moves - blanks*opp_moves

Details:
Linear combination of player's moves and opponent's moves. Coefficients depend on the number of blank spaces.
max_blanks represents the total number of cells on the board.
At the beginning of the game, "max_blanks-blanks" is small and "blanks" is high. A higher absolute coefficient is applied to the opponent's moves, which translates into a more aggressive style.
At the end of the game, blanks is lower and max_blanks-blanks is high. A higher coefficient is applied to the player's moves, which translates into a more defensive style.

I did not normalize the score by dividing by max_blanks because it should not matter. As we are comparing moves on the same level of the tree, they have the same number of blanks and max_blanks.

Result: 77.13% v.s. 74.73% (+2.39%)

### Heuristics 2 (H2): Linear combination based on blanks (defensive, then aggressive)

Implemented in linear_combination_moves_score_opposite.

Formula:
blanks*own_moves - (max_blanks-blanks)*opp_moves

Details:
Same as heuristics 1 but a higher coefficient is applied to the player's moves at the beginning of the game (which translates into a more defensive style to start) and a higher coefficient is applied to the opponent's moves at the end of the game (which translates into a more aggressive style at the end).

Result: 75.89% v.s. 74.73% (+1.16%)

Heuristics 2 is worse than Heuristics 1, so it seems better to be aggressive at the beginning and defensive at the end.
Building up on that knowledge, I created Heuristics 3.

### Heuristics 3 (H3): cutoff heuristics (aggressive, then defensive)

Implemented in cutoff_heuristics.

Formula:
At the beginning of the game (blanks > 35), be more aggressive: *own_moves - 3 * opp_moves*
At the end of the game, be more defensive: *3 * own_moves - opp_moves*

Result: 75.50% v.s. 74.73% (+0.77%)

### Heuristics 4 (H4): combination of number of available moves and players' positions on the board (edges of the board are considered better)

Implemented in moves_and_position.

Formula:
10 * (own_moves - opp_moves) + (own_dist_x + own_dist_y) - (opp_dist_x + opp_dist_y)

Details:
Based on number of moves available and each player's distance to the center of the board.
Calculates the sum of the absolute number of squares from the player's position to the center of the board along the x and y axis.
This heuristics puts a positive coefficient on the player's distance to center, pushing it towards the edges.

Result: 77.70% v.s. 74.73% (+2.96%)

As I didn't know if the center of the board was a better position than the edges, I also tried the opposite heuristics: Heuristics 5.

**Heuristics 5 (H5): combination of number of available moves and players' positions on the board (center of the board is considered better)**

Implemented in moves_and_position_opposite.

Formula:
10 * (own_moves - opp_moves) - (own_dist_x + own_dist_y) + (opp_dist_x + opp_dist_y)

Details:
Same as heuristics 4 but a negative coefficient is applied to the distance to center, therefore considering that the center of the board is better.

Result: 77.70% v.s. 74.73% (+2.80%)

Although not performing as well as Heuristics 4, this heuristics is still beating ID_Improved. I was surprised because Heuristics 4 and 5 do opposite things. Maybe having consistent decisions across moves is better than picking randomly.

## Heuristics recommendation:

My recommended heuristics is **H4: combination of number of available moves and players' positions on the board (edges of the board are considered better)**
Formula:
10 * (own_moves - opp_moves) + (own_dist_x + own_dist_y) - (opp_dist_x + opp_dist_y)

Reasons for selecting H4:

- Best score: even though it was close to other heuristics, H4 performed the best in the tournament.
- Short execution time:
    - It uses data that is accessible quickly
    - It uses fast operations

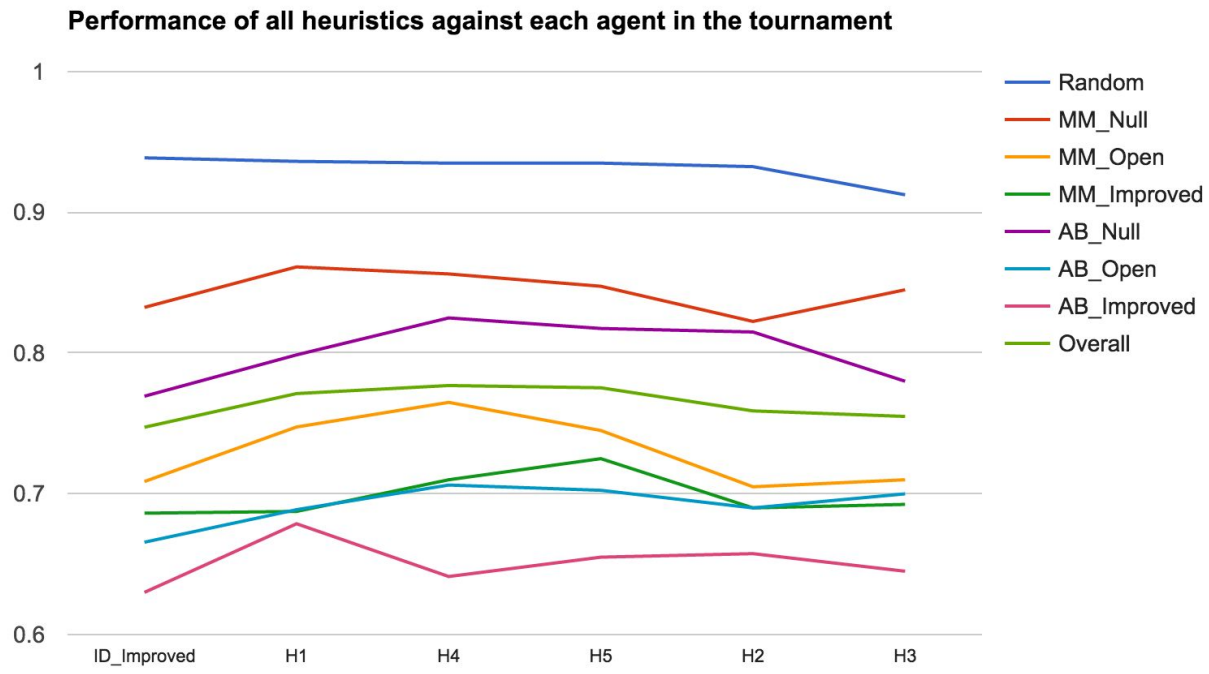
  H4 does not timeout and enables the search to go deep in the tree, leading to better results.
- H4 uses information about player and opponent: in an adversarial game, it makes sense to use information about us but also about the opponent.
- Adding information about the player's positions enables H4 to break the tie between nodes that would otherwise be considered equally good (if "own_moves - opp_moves" is the same for several nodes).

Summary of the results of all heuristics:

| | | Heuristics | | | | | |
|---|---|---|---|---|---|---|---|
| | | ID_Improved | H1 | H2 | H3 | H4 | H5 |
| Agent | Random | 93.88% | 93.63% | 93.25% | 91.25% | 93.50% | 93.50% |
| | MM_Null | 83.25% | 86.13% | 82.25% | 84.50% | 85.63% | 84.75% |
| | MM_Open | 70.88% | 74.75% | 70.50% | 71.00% | 76.50% | 74.50% |
| | MM_Improved | 68.63% | 68.75% | 69.00% | 69.25% | 71.00% | 72.50% |
| | AB_Null | 76.94% | 79.88% | 81.50% | 78.00% | 82.50% | 81.75% |
| | AB_Open | 66.56% | 68.88% | 69.00% | 70.00% | 70.63% | 70.25% |
| | AB_Improved | 63.00% | 67.88% | 65.75% | 64.50% | 64.13% | 65.50% |
| | Overall | **74.73%** | 77.13% | 75.89% | 75.50% | **77.70%** | 77.54% |

## 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:
Random < MM_Null < AB_Null < MM_Open < MM_Improved < AB_Open < AB_Improved




AB_Open is better than MM_Improved: Although Improved heuristics is better than Open heuristics, alpha-beta pruning is able to explore the tree deeper and ends up performing better than minimax.

## Battle:

In order to test my heuristics against ID_Improved in a consistent way, I created a battle mode in battle.py. I generated all the possible pairs of starting positions using the symmetry of the board. (315 different ones => 630 combinations for P1 and P2). In play_match, I'd pass the starting position and it plays 2 matches with each player starting first or second.
This was a bit helpful but it was still taking a long time and I was always testing my heuristics against ID_improved. in the tournament, it's important to win against other heuristics as well (Open etc.), not just ID_improved.

## Notes:

I'm running on a MacBook Pro, 16GB i7 4 cores.