# Analysis of heuristic functions in Isolation game

Totally 7 heuristic functions are tested. Listed below:

|  |  |
| --- | --- |
| Function name | Function implementation |
| heuristic\_diff\_score | own\_moves - opp\_moves |
| heuristic\_score\_var\_weight | own\_moves/game.move\_count |
| heuristic\_diff\_score\_with\_count | own\_moves - opp\_moves - game.move\_count |
| heuristic\_oppononet\_negtive\_score | -opp\_moves |
| heuristic\_own\_score\_fixed\_weight\_1 | 2\*own\_moves - opp\_moves |
| heuristic\_own\_score\_fixed\_weight\_2 | own\_moves/2 - opp\_moves |
| heuristic\_oppo\_score\_fixed\_weight | own\_moves - 2\*opp\_moves |

And the wining rate data of different heutistic functions listed below:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name | Function wining rate | Reference wining rate | delta |
| heuristic\_diff\_score | 66.43% | 67.86% | -1.43% |
| heuristic\_score\_var\_weight | 71.43% | 72.86% | -1.43% |
| heuristic\_diff\_score\_with\_count | 64.29% | 70.71% | -6.42% |
| heuristic\_oppononet\_negtive\_score | 65.00% | 69.29% | -3.71 |
| heuristic\_own\_score\_fixed\_weight\_1 | 64.29% | 68.57% | -4.26% |
| heuristic\_own\_score\_fixed\_weight\_2 | 75.71% | 67.86% | 7.05% |
| heuristic\_oppo\_score\_fixed\_weight | 70.71% | 66.43% | 3.28% |

The first function is actually the same with the ID\_improved function, but we can see the wining rate is somehow different. This means the wining rate is not stable even for the same heuristic in the same tournament. So if your function cannot over-perform the reference more than like 2 percent delta constantly, it is not a valid prof that your function is better.

heuristic\_score\_var\_weight and heuristic\_diff\_score\_with\_count introduces the move\_count variable which doesn’t help and it improves the calculation time, affecting the search depth.

heuristic\_oppononet\_negtive\_score simply uses the negative score of the opponent’s move, its performance is worse than ID\_improved which proves that we should both take into account of my\_move as well as opponent\_move like ID\_improved.

Moving on, how to outperform the ID\_improved? Let’s first try to improve the weight of my\_move using heuristic\_own\_score\_fixed\_weight\_1, the performance becomes worse. I think the this is expected because this will make our decision less related with the opponent’s status. Then let’s try to follow the directions from the class lecture to lower my\_move’s weight heuristic\_own\_score\_fixed\_weight\_2 or to improve the weight of the opponent’s moves (heuristic\_oppo\_score\_fixed\_weight) , the performance finally out-performs the ID\_improved, which validates that when the computer tries to get in the way of the opponents moves in an aggressive manner rather than just thinking the left moves of itself, it can yield a better result.

Since heuristic\_own\_score\_fixed\_weight\_2 which lower’s my\_move’s weight to 0.5 has the best performance, so I choose this function in the task. heuristic\_oppo\_score\_fixed\_weight which improves the weight of opponent’s moves to 2 uses the a similar idea, yet it has a smaller winning rate. It needs more deep dive to figure out why. Currently I prefer to say it is random as their performance is not stable.

From other unsatisfactory heuristics, we can learn that 1. Sometimes more variables and thus more calculation is not worthy it, because it will case the search depth shallow. 2. Just like when we are playing a game, our ultimate goal is to make our opponent has no choice rather than how many choices ‘we’ still have. If we apply this notation in the heuristics to let our opponent’s available moves less when we make choices, like we are chasing our opponent in the game, will make us win quicker and somehow avoid the “horizon effect”.