Zheng Zhang

14 February 2017

# Heuristic Analysis for Isolation Problem

1. Standard Evaluation Function

In the file sample_players.py, a simple and working evaluation function is introduced in improved_score method. It just calculates and counts the given player's moves, and the opponent's moves, then takes the difference value as the score to evaluate the goodness of the given game situation. It can be simply written as:

***score = #my_moves - #op_moves***

This evaluation is working because for a given situation, the given player has as more moves as possible, and the opponent has less moves as possible, is a good strategy, even we don't know what the result would be to the end game.

In my test runs, the agent using this score and iterative deepening search technics called ID_Improved can achieve the average result as high as 73.641%, I list the 10 times result as the following. This 73.641% winning rate in 10 times will be our baseline standard in this analysis.

| | Random | MM_ Null | MM_ Open | MM_Improved | AB_Null | AB_Open | AB_Improved | result |
|---|---|---|---|---|---|---|---|---|
| **1** | 17-3 | 16-4 | 13-7 | 16-4 | 17-3 | 14-6 | 11-9 | 74.29% |
| **2** | 18-2 | 14-6 | 16-4 | 15-5 | 16-4 | 13-7 | 14-6 | 75.71% |
| **3** | 18-2 | 15-5 | 13-7 | 17-3 | 17-3 | 14-6 | 12-8 | 75.71% |
| **4** | 16-4 | 17-3 | 12-8 | 16-4 | 13-7 | 14-6 | 11-9 | 70.71% |
| **5** | 18-2 | 17-3 | 14-6 | 11-9 | 15-5 | 10-10 | 13-7 | 70.0% |
| **6** | 17-3 | 16-4 | 16-4 | 13-7 | 17-3 | 14-6 | 10-10 | 73.57% |
| **7** | 20-0 | 15-5 | 11-9 | 14-6 | 16-4 | 16-4 | 14-6 | 75.71% |
| **8** | 18-2 | 16-4 | 14-6 | 13-7 | 17-3 | 11-9 | 10-10 | 70.71% |
| **9** | 17-3 | 18-2 | 14-6 | 13-7 | 14-6 | 13-7 | 13-7 | 72.86% |

| | Random | MM_ Null | MM_ Open | MM_Improved | AB_Null | AB_Open | AB_Improved | result |
|---|---|---|---|---|---|---|---|---|
| **10** | 19-1 | 18-2 | 15-5 | 15-5 | 17-3 | 13-7 | 11-9 | 77.14% |
| **Avg** | | | | | | | | 73.641% |

2. Custom Evaluation Functions

2.1 Non-linear rewarding with exponential

The improved score referred above as standard evaluation function is a little bit too simple, I think the rewarding to the gap between ***#my_moves*** and ***#op_moves*** should be non-linear, which means the gap is bigger, that would be much better. So this evaluation function will be called the exp score, and written as:

***score = exp(#my_moves - weight * #op_moves)***

This evaluation takes an extra parameter weight to limit how much opponent's moves would be valued, this parameter should be learned and tuned in different platform s due to different hardware resources limited search depths. I used ***1.2*** to slightly increase the goal to eliminate ***#op_moves***.

The 10 times result is recorded as below. We can find that the best and the worst case of the exp score seems are better than the improved score, and the average score is better as well. So we can say the exp score is a better eval function!

| | Random | MM_ Null | MM_ Open | MM_Improved | AB_Null | AB_Open | AB_Improved | result |
|---|---|---|---|---|---|---|---|---|
| **1** | 18-2 | 16-4 | 11-9 | 16-4 | 17-3 | 15-5 | 11-9 | 74.29% |
| **2** | 18-2 | 17-3 | 12-8 | 16-4 | 13-7 | 13-7 | 13-7 | 72.86% |
| **3** | 16-4 | 18-2 | 14-6 | 11-9 | 20-0 | 13-7 | 9-11 | 72.14% |
| **4** | 20-0 | 17-3 | 16-4 | 13-7 | 17-3 | 13-7 | 10-10 | 75.71% |
| **5** | 20-0 | 16-4 | 12-8 | 15-5 | 14-6 | 14-6 | 13-7 | 74.29% |
| **6** | 18-2 | 16-4 | 15-5 | 16-4 | 14-6 | 14-6 | 14-6 | 76.43% |

| | Random | MM_ Null | MM_ Open | MM_Improved | AB_Null | AB_Open | AB_Improved | result |
|---|---|---|---|---|---|---|---|---|
| **7** | 19-1 | 19-1 | 14-6 | 15-5 | 15-5 | 12-8 | 13-7 | 76.43% |
| **8** | 19-1 | 15-5 | 15-5 | 12-8 | 17-3 | 11-9 | 12-8 | 72.14% |
| **9** | 19-1 | 18-2 | 16-4 | 13-7 | 16-4 | 16-4 | 16-4 | 81.43% |
| **10** | 18-2 | 16-4 | 15-5 | 12-8 | 17-3 | 10-10 | 12-8 | 71.43% |
| **Avg** | | | | | | | | 74.715% |

2.2 Non-linear rewarding with polynomial

The score function is a frequently called function in search the game tree, to keep it fast would be nice, and I start to worry about how much time will exponential operation cost? If it cost too much time in iterative deepening search, we may have to search in more shallow level, is that a worth tradeoff?

To avoid exponential operation, we can use polynomial operation instead to get a cheap non-linear function. This so-called the poly score evaluation function can be written as:

***score = (#my_moves - weight * #op_moves)^2***

This function also takes ***weight*** as a parameter to tune the importance of ***#op_moves***. Again, I think no absolute good weight can be determined here, due to the same reason explained above.

Another hyper-parameter here is that we used multiply the moves difference value two times, I think this hyper-parameter should also be learned and tuned in different situation to find the best ones. However, since we want a little simplicity to run, 2 could be reasonable to use.

The 10 times results listed below. We can find that this eval function did not take the advantage in time of simple polynomial, the result is not better than our baseline.

| | Random | MM_ Null | MM_ Open | MM_Improved | AB_Null | AB_Open | AB_Improved | result |
|---|---|---|---|---|---|---|---|---|
| **1** | 18-2 | 17-3 | 13-7 | 14-6 | 14-6 | 13-7 | 10-10 | 70.71% |
| **2** | 19-1 | 17-3 | 14-6 | 12-8 | 16-4 | 13-7 | 13-7 | 74.29% |
| **3** | 19-1 | 16-4 | 14-6 | 13-7 | 13-7 | 13-7 | 11-9 | 70.71% |
| **4** | 17-3 | 19-1 | 14-6 | 13-7 | 15-5 | 15-5 | 14-6 | 76.43% |
| **5** | 16-4 | 17-3 | 15-5 | 10-10 | 16-4 | 15-5 | 12-8 | 72.14% |
| **6** | 18-2 | 18-2 | 15-5 | 14-6 | 14-6 | 12-8 | 15-5 | 75.71% |
| **7** | 18-2 | 16-4 | 14-6 | 13-7 | 15-5 | 14-6 | 9-11 | 70.71% |
| **8** | 19-1 | 16-4 | 17-3 | 16-4 | 12-8 | 15-5 | 9-11 | 74.29% |
| **9** | 19-1 | 18-2 | 13-7 | 13-7 | 17-3 | 12-8 | 11-9 | 73.57% |
| **10** | 19-1 | 17-3 | 16-4 | 12-8 | 15-5 | 10-10 | 11-9 | 71.43% |
| **Avg** | | | | | | | | 72.999% |

2.3 Local Density Evaluation Function

The above evaluation functions are evolved from the improved score, based on using ***#my_moves*** and ***#op_moves*** to evaluate how good the situation is to the given player. To understand what are other strategies can be applied to this L-shape jumping isolation game, I played several rounds with myself, and I realized the L-shape jumping make the game seem more random, because even there is a block, a proper jump can make things totally different. I also realized that to make sure the next steps will not be blocked, we want the neighbor have more spaces, we can measure how much density we have in a certain area, then use it as evaluation function. So I come up with this so called the local density score:

***score = exp(#local_blanks)***

The definition of local blanks is that we image a 5 by 5 box, the last move of the given player located in the center of the box, if it is located in corners or boundaries, we

can say that outer space is blocked, so that's why we always want to jump to the position have more space around.

I set the local area are 5 by 5, it can be 7 by 7 or else, this hyper parameter is also not determined here. I choose 5 by 5 is because that's the range of next possible move, and it shows certain influence in next coming moves.

I wonder would it work? I ran a test like above as following. We can find the test result is very close to the baseline, maybe it's not the best but we can say this local blank density works.

| | Random | MM_ Null | MM_ Open | MM_Improved | AB_Null | AB_Open | AB_Improved | result |
|---|---|---|---|---|---|---|---|---|
| **1** | 19-1 | 18-2 | 12-8 | 14-6 | 16-4 | 12-8 | 11-9 | 72.86% |
| **2** | 18-2 | 19-1 | 14-6 | 12-8 | 17-3 | 13-7 | 11-9 | 74.29% |
| **3** | 18-2 | 18-2 | 12-8 | 10-10 | 14-6 | 18-2 | 13-7 | 73.57% |
| **4** | 20-0 | 16-4 | 15-5 | 10-10 | 13-7 | 13-7 | 13-7 | 71.43% |
| **5** | 19-1 | 15-5 | 15-5 | 12-8 | 16-4 | 15-5 | 12-8 | 74.29% |
| **6** | 18-2 | 16-4 | 16-4 | 14-6 | 17-3 | 15-5 | 12-8 | 77.41% |
| **7** | 17-3 | 19-1 | 11-9 | 12-8 | 16-4 | 14-6 | 9-11 | 70.0% |
| **8** | 18-2 | 18-2 | 12-8 | 11-9 | 14-6 | 16-4 | 14-6 | 73.57% |
| **9** | 20-0 | 19-1 | 11-9 | 11-9 | 12-8 | 14-6 | 13-7 | 71.43% |
| **10** | 18-2 | 20-0 | 13-7 | 14-6 | 16-4 | 12-8 | 13-7 | 75.71% |
| **Avg** | | | | | | | | 73.456% |

2.4 Boost Evaluation Function

What do we know so far? The improved score is baseline, the exp score has the best performance so far, the polynomial score not that good, and the local density score provides another idea of strategies.

Maybe we can put the exp score and the local density score together, to eval the given game situation based on different criteria. We can give the definition of the boost score as:

***score = weight1 * exp_score + weight2 * local_density_score***

After some trial runs on my machine, I put **weight1** equal to 0.7 and **weight2** equal to 0.3 to balance a good result. The last result is listed below. We can find the result is not as good as exp score, but is slightly better than base line.

| | Random | MM_ Null | MM_ Open | MM_Improved | AB_Null | AB_Open | AB_Improved | result |
|---|---|---|---|---|---|---|---|---|
| **1** | 20-0 | 19-1 | 13-7 | 9-11 | 16-4 | 15-5 | 11-9 | 73.57% |
| **2** | 18-2 | 16-4 | 14-6 | 14-6 | 15-5 | 15-5 | 15-5 | 76.43% |
| **3** | 18-2 | 16-4 | 12-8 | 14-6 | 19-1 | 14-6 | 12-8 | 75% |
| **4** | 16-4 | 16-4 | 15-5 | 13-7 | 17-3 | 14-6 | 13-7 | 74.29% |
| **5** | 16-4 | 14-6 | 14-6 | 16-4 | 16-4 | 11-9 | 10-10 | 69.29% |
| **6** | 18-2 | 16-4 | 12-8 | 13-7 | 14-6 | 15-5 | 13-7 | 72.14% |
| **7** | 20-0 | 15-5 | 10-10 | 13-7 | 14-6 | 16-4 | 13-7 | 72.14% |
| **8** | 16-4 | 16-4 | 14-6 | 12-8 | 17-3 | 13-7 | 14-6 | 72.86% |
| **9** | 19-1 | 16-4 | 17-3 | 14-6 | 14-6 | 15-5 | 13-7 | 77.14% |
| **10** | 20-0 | 18-2 | 15-5 | 11-9 | 15-5 | 13-7 | 12-8 | 74.29% |
| **Avg** | | | | | | | | 73.715% |

3. Conclusion

To choose the best evaluation function, Let’s see more statistic data to support our last decision.

I first calculated the winning rates of the exp score, the local density score and the boost score when they against different opponents.

| | Random | MM_Null | MM_Open | MM_Improved | AB_Null | AB_Open | AB_Improved |
|---|---|---|---|---|---|---|---|
| **1** | 92.5% | 84% | 70% | 69.5% | 80% | 65.5% | 61.5% |

(the exp score)

| | Random | MM_Null | MM_Open | MM_Improved | AB_Null | AB_Open | AB_Improved |
|---|---|---|---|---|---|---|---|
| **1** | 92.5% | 89% | 65.5% | 60.0% | 75.5% | 71.0% | 60.5% |

(the local density score)

| | Random | MM_Null | MM_Open | MM_Improved | AB_Null | AB_Open | AB_Improved |
|---|---|---|---|---|---|---|---|
| **1** | 90.5% | 81% | 68% | 64.5% | 78.5% | 70.5% | 63% |

(the boost score)

Then we can calculate their expectation and standard derivation as follow:

| | Expectation Winning Rate | Standard Derivation |
|---|---|---|
| **The Exp Score** | 0.747 | 0.640 |
| **The Local Density Score** | 0.734 | 0.672 |
| **The Boost Score** | 0.737 | 0.627 |
| | | |

Ideally, we want our evaluation function can achieve the highest expectation winning rate, and lowest standard derivation, which means when playing against a new opponent, the agent has higher probability to win, and lower probability to get a extremely result, or more stable.

The exp score obviously has the highest winning rate, while the boost score has the lowest standard derivation, so we have to choose our recommended evaluation function between them.

I would give the boost score to the last recommendation.

1. The boost score is a combination of different strategies, which means it is more powerful to play against other players with different strategies. The exp has

highest winning rate, but it might only apply to certain kinds of opponent players. The standard derivation explained the boost score is more regularized than others.

2. The boost score got more parameters to tune, simply because it is combination of different scores, more parameters to tune is good, because to solve a complicated problem like Isolation Game, the space of solves are usually located in high dimension, more parameters will reduce the chance we overfit our solution to certain scenarios.

3. The boost score is scalable, which means if we come up a new strategy, and it will not exponentially increase searching time, we can just append it to the boost score, make a new evaluation function, and expect that give us a better result. So the boost score is actually a good framework to work with.