Heuristic Analysis for Isolation Game

John Carpenter

March 1, 2017

1 Isolation Analysis

For this isolation game, a game agent was built that was able to defeat a variety of different computer opponents roughly 85% time. The game agent used a combination of minimax and alphabeta pruning to search through the game space for the correct moves.

1.1 Discussion

Performance
After running the tournament a number of times one of the important factors in the outcome was the depth of search that the program was able to complete. Turns must be completed in 150ms or the game was forfeited with a loss. To this end the performance of the `get_move` determined how often the program was able to win. Optimizing this method was important to a successful run. While there wasn't any global metrics collected a cursory calculation showed that the number of searches that went to full-depth was less than 40% and those were likely near the end of the match only

In creating the game agent we implemented a tree-node architecture with the `GameNode.py` class. That architecture allowed for flexible tree configurations and separated the construction of the tree with it's scoring/pruning functions. However, considering the unittest were written first we had to modify some of the code to function within the parameters and it lost some of its effectiveness.

It would be worthwhile to continue some optimization on the `get_move` in order to see it's impact on the solution. The first step might be collecting better metrics on the performance of the search and scoring routines and profiling the method calls in more depth.

1.2 Heuristics

In order to evaluate the board state at each we created 4 heuristics as a measure of the quality of the board. The goal of this section was to arrive at a

heuristic that was able to statistically outperform the `ID_Improved` model that was provided.

- Heuristic 1 - Baseline Low Performance
- Heuristic 2 - Moves Left (`ID_Improved` model)
- Heuristic 3 - Warnsdorf's rules
- Heuristic 4 - Modified Moves Left

Notes: The results and analysis here are based upon a limited run of a couple hundred game instances. As shown in the sections below there is a large variation in the winning percentages. Given the variability and small sample size the results may not be statistically significant

The details of each of the repective measures are in their sections below.

1.2.1 Heuristic 1 - Baseline Low Performance

The first heuristic was done to provide a baseline for the remaining test cases. This should allow us to understand how much a change in the scoring heuristic is expected to impact the testing performance.

Results

As expected, the results from the first heuristic performed worse than the `ID_Improved` model.

```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random     Result: 19 to 1
  Match 2: ID_Improved vs   MM_Null    Result: 18 to 2
  Match 3: ID_Improved vs   MM_Open    Result: 13 to 7
  Match 4: ID_Improved vs MM_Improved   Result: 17 to 3
  Match 5: ID_Improved vs   AB_Null     Result: 18 to 2
  Match 6: ID_Improved vs   AB_Open    Result: 18 to 2
  Match 7: ID_Improved vs AB_Improved  Result: 17 to 3

Results:
----------
ID_Improved         85.71%

*************************
```

```
   Evaluating: Student
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random     Result: 17 to 3
  Match 2:   Student   vs   MM_Null    Result: 16 to 4
  Match 3:   Student   vs   MM_Open    Result: 13 to 7
  Match 4:   Student   vs MM_Improved  Result: 13 to 7
  Match 5:   Student   vs   AB_Null    Result: 17 to 3
  Match 6:   Student   vs   AB_Open    Result: 13 to 7
  Match 7:   Student   vs AB_Improved  Result: 10 to 10

Results:
----------
Student             70.71%
```

1.2.2 Heuristic 2

The second heuristic was identical to the `ID_Improved` model. We ran this model to demonstrate some of the variability in the win %. Even within the tests in this document we see the range of win % from the `ID_Improved` vary between 82% and 90%. **Results**

```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random     Result: 20 to 0
  Match 2: ID_Improved vs   MM_Null    Result: 20 to 0
  Match 3: ID_Improved vs   MM_Open    Result: 17 to 3
  Match 4: ID_Improved vs MM_Improved  Result: 15 to 5
  Match 5: ID_Improved vs   AB_Null    Result: 20 to 0
  Match 6: ID_Improved vs   AB_Open    Result: 16 to 4
  Match 7: ID_Improved vs AB_Improved  Result: 15 to 5


Results:
----------
ID_Improved         87.86%

*************************
   Evaluating: Student
```

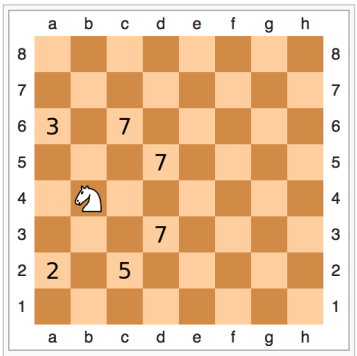
```
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random     Result: 19 to 1
  Match 2:   Student   vs   MM_Null    Result: 20 to 0
  Match 3:   Student   vs   MM_Open    Result: 18 to 2
  Match 4:   Student   vs MM_Improved  Result: 17 to 3
  Match 5:   Student   vs   AB_Null    Result: 19 to 1
  Match 6:   Student   vs   AB_Open    Result: 18 to 2
  Match 7:   Student   vs AB_Improved  Result: 17 to 3


Results:
----------
Student             91.43%
```

1.2.3 Heuristic 3

The third heuristic attempted to utilize a slightly different approach to scoring the model.
There is a mathematical problem, called the Knights Tour that is relevant to this game. In this problem the goal is to move a Knight around a chessboard such that you touch every point on the board without visiting the same point twice. In this problem, the mathematician H.C Warnsdorf determined that the solution involves always moving to the point with the fewest number of connected locations. It is called Warnsdorf's rule.
In order to implement a variant of that form, we want the program to choose a location with the minimum number of moves (greater than 0), and the opponent to choose the move with the most amount of moves. (Although the second part isn't as crucial to the formula). We based the quality of the board as:



A graphical representation of Warnsdorf's Rule. Each square contains an integer giving the number of moves that the knight could make from that square. In this case, the rule tells us to move to the square with the smallest integer in it, namely 2.

```
if(player_moves_left >= 2):
    return 8 - float(player_moves_left)
else:
    return -1
```

https://en.wikipedia.org/wiki/Knight's˙tour##Warnsdorf.27s˙rule

Results

```
*************************
 Evaluating: ID_Improved
*************************

Playing Matches:
----------
  Match 1: ID_Improved vs   Random     Result: 19 to 1
  Match 2: ID_Improved vs   MM_Null    Result: 18 to 2
  Match 3: ID_Improved vs   MM_Open    Result: 18 to 2
  Match 4: ID_Improved vs MM_Improved  Result: 17 to 3
  Match 5: ID_Improved vs   AB_Null    Result: 19 to 1
  Match 6: ID_Improved vs   AB_Open    Result: 17 to 3
  Match 7: ID_Improved vs AB_Improved  Result: 17 to 3


Results:
----------
ID_Improved         89.29%

*************************
   Evaluating: Student
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random     Result: 19 to 1
  Match 2:   Student   vs   MM_Null    Result: 19 to 1
  Match 3:   Student   vs   MM_Open    Result: 16 to 4
  Match 4:   Student   vs MM_Improved  Result: 11 to 9
  Match 5:   Student   vs   AB_Null    Result: 19 to 1
  Match 6:   Student   vs   AB_Open    Result: 14 to 6
  Match 7:   Student   vs AB_Improved  Result: 13 to 7


Results:
----------
Student             79.29%
```

1.2.4 Heuristic 4

```
  *************************
   Evaluating: ID_Improved
  *************************
```

```
Playing Matches:
----------
  Match 1: ID_Improved vs   Random     Result: 17 to 3
  Match 2: ID_Improved vs   MM_Null    Result: 19 to 1
  Match 3: ID_Improved vs   MM_Open    Result: 16 to 4
  Match 4: ID_Improved vs MM_Improved  Result: 13 to 7
  Match 5: ID_Improved vs   AB_Null    Result: 19 to 1
  Match 6: ID_Improved vs   AB_Open    Result: 16 to 4
  Match 7: ID_Improved vs AB_Improved  Result: 16 to 4


Results:
----------
ID_Improved         82.86%

*************************
   Evaluating: Student
*************************

Playing Matches:
----------
  Match 1:   Student   vs   Random     Result: 16 to 4
tournament.py:100: UserWarning: One or more agents lost a match this round due to timeout.
  warnings.warn(TIMEOUT_WARNING)
  Match 2:   Student   vs   MM_Null    Result: 18 to 2
  Match 3:   Student   vs   MM_Open    Result: 16 to 4
  Match 4:   Student   vs MM_Improved  Result: 18 to 2
  Match 5:   Student   vs   AB_Null    Result: 20 to 0
  Match 6:   Student   vs   AB_Open    Result: 17 to 3
  Match 7:   Student   vs AB_Improved  Result: 17 to 3


Results:
----------
Student             87.14%
```