Heuristic report for AIND-Isolation algorithm

Matei Stefan Neagu

Introduction

The aim of the report is to present an analysis on 3 heuristics that have been tried for the Isolation project in the Artificial Intelligence Nanodegree. The remainder of the report presents the techniques used to evaluate different heuristics, the 3 proposed heuristics and recommendations for picking the best method.

Comparison methods

For the comparison of 2 heuristics the following steps were taken:

1. Using the tournament.py script, 2 players were initialized with each heuristic as an evaluation function
2. The comparison part in the tournament.py script was run 10 times
3. The performance of the heuristics were evaluated using:
   - **Average performance** for each heuristic. The average percentage of winnings for the 10 runs was calculated for each heuristic.
   - **P-value** of a two-sample t-test test on the performances of the heuristics. The p-value of a Student two-sample test calculates the probability that 2 populations come from normal distributions with the same average. In this case the 2 populations used are the percentages of games won by each player in each run. Basically what is measured is the probability that the differences observed in the average performances are false.

Heuristic1: Varied weights on the number of adversary moves

The easiest way in which a new heuristic can be created from the **improved_score()** heuristic is to change the weight of the number of opponent moves in the final score. The heuristic obtained this way will be called **weighted_score()**.

It turns from:

Number_own_moves - number_of_opponent_moves

to:

Number_own_moves - w*number_of_opponent_moves

with **w** chosen by the user.

Different values have been tried for **w** both <1 which relaxes the need to block opponent's moves and >1 which strenghtens it. Figure 1 visually presents the means and standard deviations of the perfomances of the **improved_score()** and the **weighted_score()** heuristics for different weights.

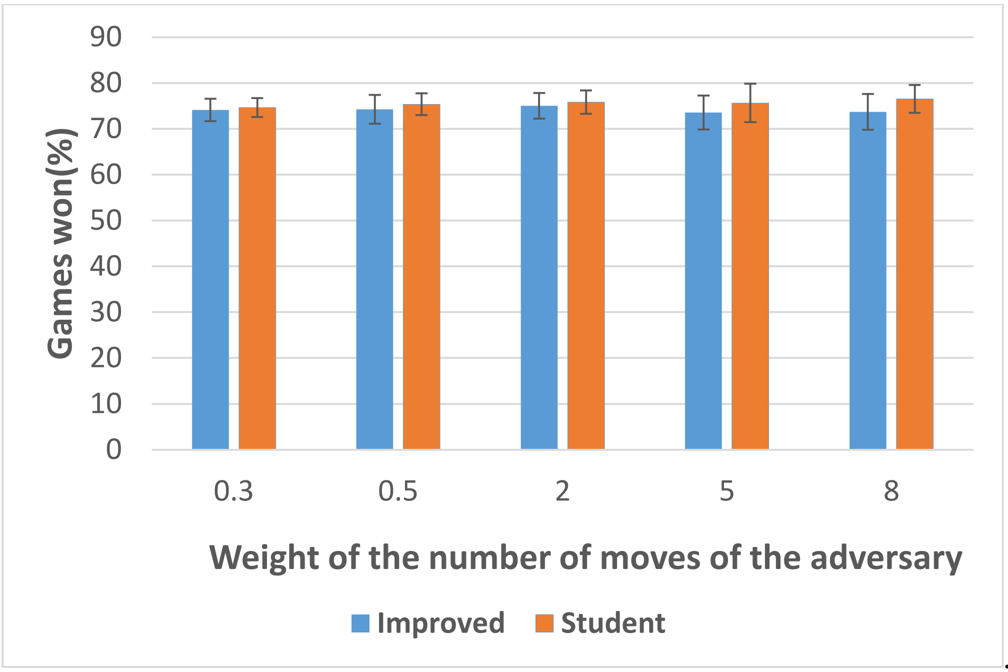

Figure1: Percentage of games won by the 2 heuristics for different weights. **Improved** represents the score for the player using **improved_score()** heuristic while **Student** presents the score of the player using **weighted_score()** as a heuristic

As it can be observed both in Figure1 and Table1, more aggressive behaviour of the player seems to lead to better results. The maximum weight used was 8 because the number of possible moves for a player in a game is 8. This means that when an action that can block 1 move of the opponent exists, it will always be picked. A weight of 8 seems to create the highest average performance and also the p-value obtained is quite low. Although a p-value of 0.1 is over the generally accepted 0.05 it still means that there is a 90% probability that the difference observed is real.

Taken this into consideration the heuristic:

Number_own_moves-8*number_of_opponent_moves

was picked as the best solution.

Weight of opponent moves	Average score (%) **improved_score()**	Average score (%) **weighted_score()**	P-value
0.3	74.142	74.643	0.6257
0.5	74.284	75.358	0.3611
2	74.999	75.857	0.1825

5	73.572	75.642	0.5969
8	73.714	76.501	0.1005

Table1: Performance for different weights on the opponent moves

Heuristic2: Open space

While the **improved_score()** heuristic does a good job at precisely evaluating the current situation on the board by using the player's and opponent's possible moves, it does a poor job at predicting the future of the game. This happens because of the way a horse moves. A position that seems good at a certain moment(has lots of possible follow-ups) can easily lead to a position with very few possible moves. In the same way, a position with even one move could lead to a position with many possible moves.

An intuitive solution to this problem would be to move the horse to an area with many open moves, which although does not guarantee that many possible moves will be available in the future, it provides an approximation for it. A visual explanation for the method and its implementation are provided in Annex A. The method will be called **open_space()** in the remainder of the report.

Figure 2 presents a comparison between the performance of the **open_space()** method and **improved_score()**. On average **improved_score()** method won **75.49%** of the games while the **open_space ()** one won just **73.14%** of the games. The p-value of difference in performance was 1.4.

Although **open_space()** gives poorer results than **improved_score(),** it leads to quite a high winning rate, qualifying it as a viable heuristic for the game.

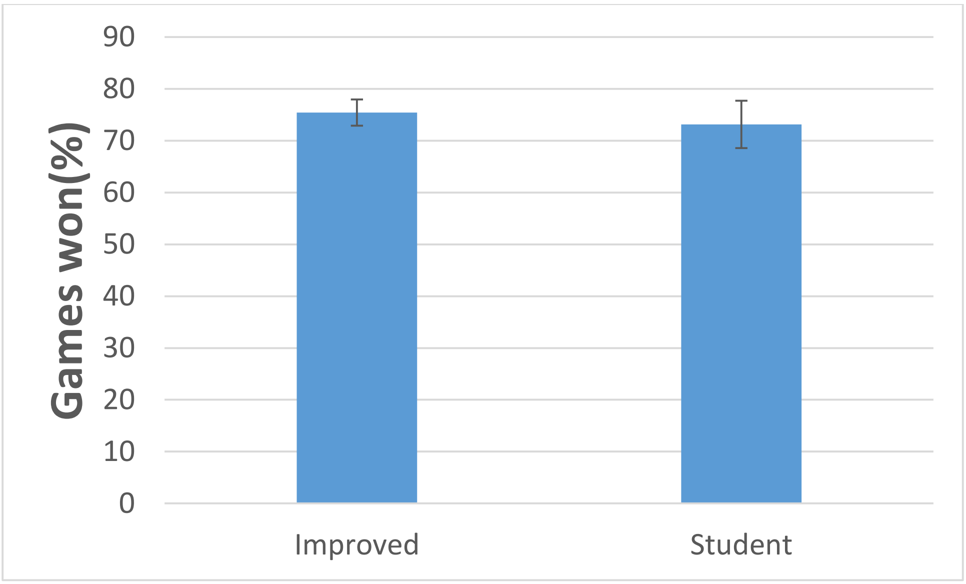

Figure2: Percentage of games won by the 2 heuristics for different weights. **Improved** represents the score for the player using **improved_score()** heuristic while **Student** presents the score of the player using **open_space()** as a heuristic

Heuristic3: Hybrid

An ideal heuristic would have the capacity to evaluate the future the way **open_space()** does but also precisely evaluate the present as **improved_score()**. This can be created with a combination of the 2 methods which will be called **hybrid_score()**. In order to optimise performance **weighted_score**() with a weight of 8 was used instead of **improved_score().** The new heuristic to evaluate is :

Number_own_moves-8*number_of_opponent_moves-open_space()

Figure 3 presents a comparison between the performance of **hybrid_score()** and **improved_score()**. As it can be observed there is a clear increase in the average performance from 73.64% to 78%. Also the p-value for the test was 0.0003 showing that the difference is statistically significant.

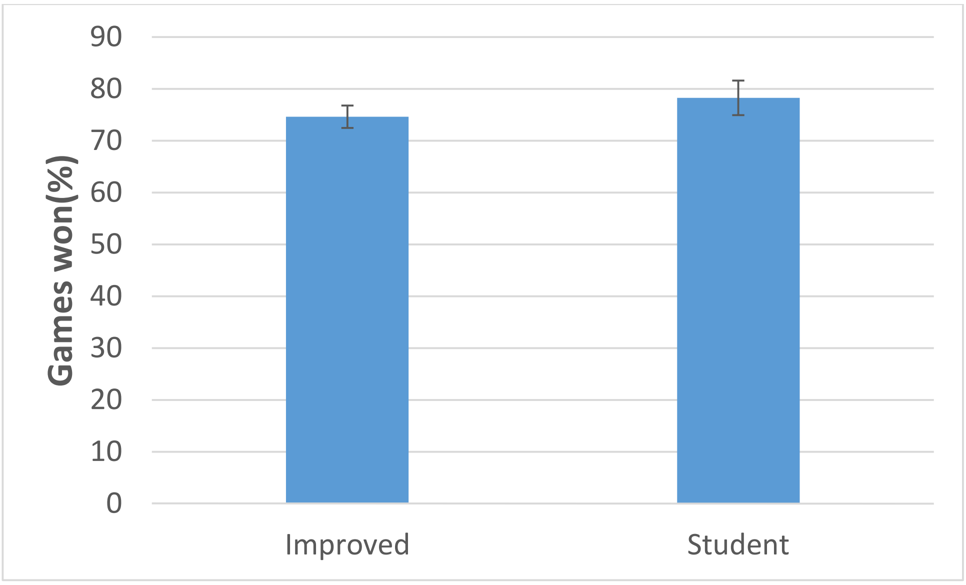

Figure3: Percentage of games won by the 2 heuristics for different weights. **Improved** represents the score for the player using **improved_score()** heuristic while **Student** presents the score of the player using **hybrid_score()** as a heuristic

In order to test if the heuristic can be easily improved, different weights were tested for the **open_space()** element in the heuristic. The heuristic is now :

Number_own_moves-8*number_of_opponent_moves-w*open_space()

2 values were tested for w against the simple hybrid heuristic, 0.5 and 2 and in both cases the differences were statistically insignificant with p-values of around 0.7. As no significant improvements appeared, the initial hybrid heuristic was further used.

While it gives better performance, the **hybrid_score()** heuristic has one disadvantage compared to the **improved_score()** heuristic which is the fact that it takes longer time. This can create problems if the difference in time between the 2 heuristics could lead to a time-out. There is only one call of the heuristic between 2 checks of the remaining time which means that the difference in execution time between the 2 heuristics represents the time added in which a timeout could appear.

In order to check the average timings, after 15 moves have taken place on the board, the heuristics are called 10000 times to evaluate the board and the maximum execution time for each heuristic is evaluated. With this test for **hybrid_score()** a maximum execution time of 0.3 ms was obtained while for improved score a maximum time of 0.14 ms was obtained. The difference seems to be of around 0.2 ms. This could create problems if the remaining time after a round passes is close to 0.5-1 ms.

To test that how close the period of execution of the program is to the round limit, 100 games with 15 moves each were played with players using **hybrid_score()** as a heuristic and the time of each move was saved, thus having 1500 rounds whose timing was recorded. The players had a time limit of 200 ms per round and the timer threshold was set to 10 ms. Then the same was done for 2 players using **improved_score()** as a heuristic.

The maximum timings for a round were 191.41 ms for **improved_score()** and 190.35 ms for **the hybrid_score()**. This points out that there is no danger in using the **hybrid_score()** heuristic as more than 8 ms which is a lot higher than the 0.2 ms added remained no matter what heuristic was used.

The average timings for the 2 cases were 190.07 ms per round for the **improved_score()** and 190.12 ms for the **hybrid_score()**. In order to make sure that this does not significantly affect the performance a test was done on a player using **improved_score()** as a heuristic and 10 ms as a timeout threshold and another player using **hybrid_score()** as a heuristic and 10.05 ms as a timeout threshold. The results were an average performance of 75.5% for the improved score player and an average of 77.35% for the player using **custom_score()**. The p-value for the performance was 0.22. Although the difference was not statistically significant, it should be noted that the improved score did not give an average performance of more than 76% in any of its runs which indicates the **hybrid_score()** algorithm as the more performant one even with the handicap given by the longer timeout.

In order to make a final check to see if the differences seen in performance are real, a simulation with 400 repetitions of the comparison part in tournament results was done. The players were a player using **improved_score()** as a heuristic against a player using hybrid_score with a 10.05 ms timeout. The average winning percentage was 74.7% for the player using **improved_score()** and 77.5% for the player using **hybrid_score()**. The p-value was $5*10^{-33}$ so it can be stated with confidence that the hybrid heuristic provides an improvement in the algorithm performance of almost 3%.

Recommendations on picking the heuristic

Out of the 3 heuristic proposed, the one recommended for use is **hybrid_score()**. The reasons to do so are:

a) It is the only one to provide a clearly increased performance compared to the **improved_score()** heuristic
b) Although it increases the execution time of the algorithm the increase is not significant enough to make the player lose the game due to a timeout or to lose performance if the timeout threshold is increased in accordance to the difference in timing
c) It combines the advantages of **open_space()** as well as **improved_score()** heuristics
d) It is simple to implement and intuitive which could establish it as a popular heuristic for the problem

Annex A The open_space() heuristic

If the player is in a situation like the one presented in Figure A1, the heuristic will make it move to the area A7-D4.

The way this can be evaluated on a move which have been done already is to split the table in 4 areas relative to the new position. Assuming this would be C6, the 4 areas would be A7-C6, D6-G7, D5-G1, and A1-C5. The player will know that the move was made in the area with the most free spaces by comparing the number of open spaces in each of the 4 areas. In case the move was done in the area with more spaces, the 4 areas will have a more even number of spaces. If the player moves away from it, for example move to E2, one area, in this case A2-E7 will have a lot more free spaces than the others.

The difference in the number of open spaces in the four areas can be evaluating by computing their standard deviation(std). One area with way more open spaces than the other 3 will create a higher std while quite even numbers of open spaces between areas will create a lower std.

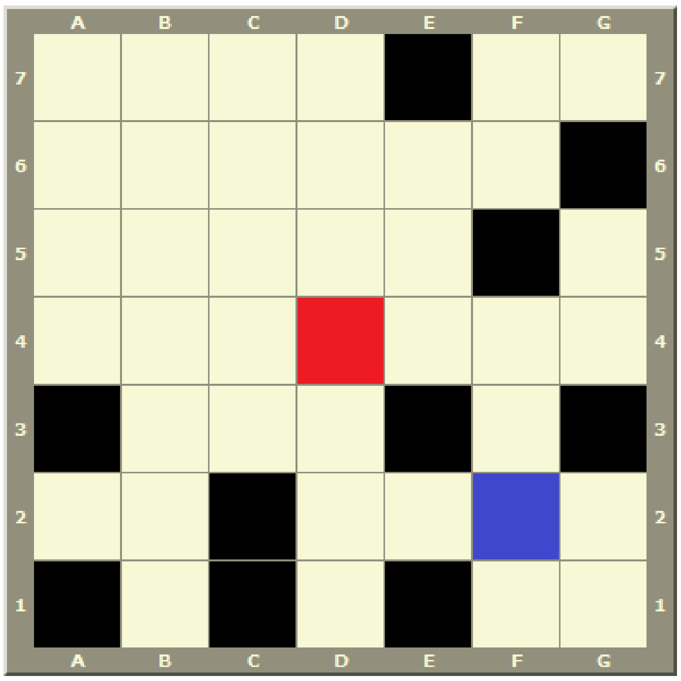

Figure A1 Representation of a hypothetical situation in order to illustrate the **open_space()** heuristic. Red square represents the current position of the player, the blue one the current position of the opponent, the light yellow square positions that were not passed through and black squares positions that have been occupied during the game.

The algorithm is as follows:

- For the player's current position split the table in 4 areas
- Compute the number of open positions for each area
- Return the negative of the standard deviation of the number of open spaces of the areas as high standard deviations should be penalized