Evaluation of Different Heuristic Value Functions

Safak Ozkan
shafaksan@gmail.com

1. INTRODUCTION

The purpose of this project is to build an AI agent that plays a game of Isolation.The Isolation is a board game played on a checker board with one piece per player, where the players move their pieces L-shaped moves, the same way a knight moves on a chess board.The positions visited by the pieces gets blocked for future moves.The player who doesn't have any available moves left loses the game.

In this project 3 separate heuristic value functions have been devised to estimate the value of the score at any truncated node of the search tree.These heuristic functions are created by trial-and-error and were tested against 7 different AI programs each with their own algorithms and heuristic value functions in a tournament of 10 Isolation games.

CPU opponent 1, Random: An agent that randomly chooses a move each turn.

CPU opponent 2, MM_Open: Fixed-depth minimax search with `open_move_score` heuristic

CPU opponent 3, MM_Center: Fixed-depth minimax search with `center_score` heuristic

CPU opponent 4, MM_Improved: Fixed-depth minimax search with `improved_score` heuristic

CPU opponent 5, AB_Open: Fixed-depth alpha-beta search with `open_move_score` heuristic

CPU opponent 6, AB_Center: Fixed-depth alpha-beta search with `center_score` heuristic

CPU opponent 7, AB_Improved: Fixed-depth alpha-beta search with `improved_score` heuristic

The game playing agent is designed to search the game tree and return the move that has the highest evaluation score, using iterative-deepening and alpha-beta pruning that

There are also 3 heuristic value functions implemented by the AI opponent:

1. `open_move_score`: returns the number of moves available to the player.
2. `center_score`: returns the square of the distance to the
3. `improved_score`:The difference between the number of moves available to the player and its opponent.

2. HEURISTIC FUNCTIONS

The Intuition behind designing of a heuristic function is important.The first clear idea is to calculating the number of moves available to our AI player. However, this is a naive approach that fails to reflect the two-player competitive nature of the game. Instead, a heuristic function that computes the difference between the available moves of the player and its opponent seems to reflect the nature of the game better.

All the heuristic value functions were based on the thought that central region of the board is safer than the edges or the corners. For this reason, a *directional weight* function was created based on the horizontal and vertical distances of the current position to the center of the board. For a cell in the center of any of the four edges, the directional weight score is 0.5 and it increases linearly to 1.0 at the center. The final *directional weight* is calculated as the product of the horizontal and vertical components of the weights, such that, the center cell will have a weight score of 1.0*1.0=1.0, and the corner cells will have a weight score of 0.5*0.5=0.25.

0.2	0.3	0.4	0.5	0.4	0.3	0.2
0.3	0.4	0.6	0.7	0.6	0.4	0.3
0.4	0.6	0.7	0.8	0.7	0.6	0.4
0.5	0.7	0.8	1.0	0.8	0.7	0.5
0.4	0.6	0.7	0.8	0.7	0.6	0.4
0.3	0.4	0.6	0.7	0.6	0.4	0.3
0.2	0.3	0.4	0.5	0.4	0.3	0.2

Figure 1. Directional weights for the positions on a 7-by-7 board.

```
def get_directional_weights(location,width,height):
    x,y = location
    xo_idx = (width+1)/2-1
    yo_idx = (height+1)/2-1
    wx = 1 - abs(x - xo_idx)/((width-1)/2) * 0.5
    wy = 1 - abs(y - yo_idx)/((height-1)/2) * 0.5
    return wx*wy
```

Code snippet 1. This helper function takes the cell location and board dimensions as input and calculates directional weights and returns their product.

- Heuristic One: `custom_score`

The first heuristic value function `custom_score` calculates a simple value: the number of available moves weighted by the directional weight of its current position. Since this function makes only one call to the central score function it's fast at runtime possibly allowing the Iterative Deepening algorithm to search down more plies.

- Heuristic Two: `custom_score_2`

The second heuristic function `custom_score_2` computes the difference of the number of moves available to each players scaled by the corresponding directional-weight scores of the current positions of the pieces.

- Heuristic Three: `custom_score_3`

The third heuristic function `custom_score_3` computes the difference of the sum of the directional-weight scores for the positions of all legal moves of the AI agent and its opponent. This can be thought of as an improved version of directional-weighted scores for the legal moves of the CPU and the AI agents separately. The heuristic function `custom_score_2` is computationally more expensive than the heuristic `custom_score_3`, as it calculates the directional weight of each legal move.

```
def custom_score(game, player):
    my_legal_moves = game.get_legal_moves(player)
    curr_loc = game.get_player_location(player)
    my_weight = get_directional_weights(curr_loc, game.width, game.height)
    return float(len(my_legal_moves)) * my_weight
```

Code snippet 2. Heuristic function `custom_score` .

```
def custom_score_2(game, player):
    my_legal_moves = game.get_legal_moves(player)
    my_weight = get_directional_weights(game.get_player_location(player),\
game.width, game.height)
           opponent_legal_moves = game.get_legal_moves(game.get_opponent(player))
    opponent = game.get_opponent(player)
    opponent_weight = get_directional_weights(\
             game.get_player_location(opponent), game.width, game.height)
    return float(len(my_legal_moves)) * my_weight -\
           float(len(opponent_legal_moves)) * opponent_weight
```

Code snippet 3. Heuristic function `custom_score_2`.

```
def custom_score_3(game, player):
    my_legal_moves = game.get_legal_moves(player)
    opponent_legal_moves = game.get_legal_moves(game.get_opponent(player))

    my_score = sum([get_directional_weights(move, game.width, game.height)\
       for move in my_legal_moves])
    opponent_score = sum([get_directional_weights(move, game.width,\
                    game.height) for move in opponent_legal_moves])
    return my_score - opponent_score
```

Code snippet 4. Heuristic function `custom_score_3`.

3. Results and Evaluation

The performance of the AI agent with alpha-beta pruning, iterative deepening and its custom heuristic value functions are tested against the CPU agents that employed a Random agent, a Minimax agent and an alpha-beta pruning agent with open, center and improved heuristic functions in a tournament of 10 games round.

The forfeiting of the game was prevented by randomly selecting a legal move before the search algorithm was initiated. Hence, the outcome of the tournament has a degree of randomness.

Designing a reliable and consistent heuristic value function requires extensive domain knowledge. Even if a heuristic function with a good evaluation accuracy might be attainable, it's also highly likely that it would have a high complexity which would make it too burdensome and costly to evaluate at runtime. Hence, the tradeoff between the accuracy of the heuristic and its computational cost might limit the AI agent's ability to search deeper levels of the game tree during Iterative Deepening Search, hence making the heuristic function undesirable.

The heuristics developed in this project have only shown mild signs of better performance over the default AB_Improved heuristic. Nevertheless, the second heuristic function often performed roughly about 5% better than the default improved heuristic and the other two custom heuristic functions (cf. Table 1).

```
        Match #    Opponent    AB_Improved  AB_Custom   AB_Custom_2  AB_Custom_3
                                Won|Lost     Won|Lost    Won|Lost     Won|Lost
           1        Random       9  | 1     10  | 0     10  | 0     10  | 0
           2        MM_Open      9  | 1      9  | 1      9  | 1      7  | 3
           3       MM_Center    10  | 0      9  | 1      9  | 1      8  | 2
           4      MM_Improved    5  | 5      8  | 2      8  | 2      8  | 2
           5        AB_Open      5  | 5      6  | 4      9  | 1      5  | 5
           6       AB_Center     6  | 4      7  | 3      8  | 2      6  | 4
           7      AB_Improved    7  | 3      5  | 5      6  | 4      6  | 4
--------------------------------------------------------------------------
                  Win Rate:      72.9%        77.1%        84.3%        71.4%
```

Table 1. The results of the tournament.