Heuristics Analysis

Ryan Shrott

Heuristic 1: Look Ahead (Aka Double Jumps)

This heuristic computes the difference in the available double jumps available to the agent minus that of its opponent. I postulate that a double jump is more telling of a descent move than that of a single jump since it foretells information about the subsequent moves that may be available to the agent. The downside of the heuristic is the loss in computational efficiency. In particular, we must call the get_moves() function from each move computed by the get_legal_moves() function. So if there are n legal moves available, we will make n + 1 calls to the get_legal_moves function. Therefore, the time complexity moves from constant time to linear time compared to the simple improved heuristic. It is not obvious whether the additional information gained by using the Look Ahead heuristic is worth the loss in computational efficiency.

Heuristic 2: Corner Penalty

This heuristic is a simple extension of ID_Improved which penalizes moves which move to corners. I postulate that a corner move may cause the agent to get trapped and should therefore be penalized. The added time complexity is a constant factor. Therefore, I expect that this heuristic should perform better than that of ID_Improved.

Heuristic 3: Centre Preference

This heuristic is also an extension of ID_Improved which penalizes moves which are further from the centre of the board. The idea is that moves closer to the centre often have a strong positional advantage and should be weighted higher in the computation of the available moves. We only consider this advantage in the calculation if the number of moves available to the agent and to its opponent are equal. In that case, we compute the opponents Manhattan distance to the centre minus that of the agents Manhattan distance to the centre. We also normalize to ensure that we are never any better than having more moves available than that of the opponent (or vise versa for that of the agent). The added complexity here is a constant factor. Therefore, I expect that this heuristic should perform better than that of ID_Improved.

Calibration Procedure: (gentic.py)

All of the discussed heuristics contain constant factors which need to be optimized. As an example, consider a simple weighting of the ID_Improved heuristic:

f(w) = w * num_my_moves - (1-w) * num_opponent_moves, where w in [0,1]

The optimization process was designed using an evolutionary/genetic programming approach. The basic approach is to create a random population of agents (say, 100) (varying the weights) and let them play against each other 1000 times. Let the parents be the top performing agents with some poorer performing agents mixed in for genetic diversity. Randomly breed some parents to create children.

Breeding process: We define a child's weight as follows:

childWeight = random.randrange(father.w*0.95, mother.w*1.05).

We adjust for bordering cases in the obvious way. The new population is formed by the parents and the newly created children. There is also a non-trivial mutation process which helps with genetic diversity and eliminates local maxima convergence. The evolution cycle is repeated until the population converges to a global maxima. We invite the reader to view the code (genetic.py) for all the details of this implementation.

Results

```
                    *************************
                        Playing Matches - Ryan Shrott 2017-07-24
                    *************************

Match #   Opponent    AB_Improved   look_ahead     corner       centre     sqaure_diff      divis
                       Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost   Won | Lost
    1       Random      19  |   1    17  |   3    19  |   1    18  |   2    18  |   2    19  |   1
    2       MM_Open     17  |   3    16  |   4    17  |   3    13  |   7    11  |   9    16  |   4
    3      MM_Center    18  |   2    19  |   1    17  |   3    17  |   3    15  |   5    18  |   2
    4     MM_Improved   14  |   6    17  |   3    15  |   5    14  |   6    17  |   3    16  |   4
    5       AB_Open      8  |  12    12  |   8    10  |  10    11  |   9    10  |  10    12  |   8
    6      AB_Center    11  |   9    12  |   8    12  |   8    13  |   7     9  |  11     8  |  12
    7     AB_Improved   11  |   9    13  |   7    10  |  10     9  |  11     9  |  11    11  |   9
-------------------------------------------------------------------------------------------------
          Win Rate:      70.0%        75.7%        71.4%        67.9%        63.6%        71.4%
```

As shown above, the look_ahead and corner heuristics outperform the AB_Improved heuristic function. The Centre Preference heuristic does not seem to perform better than the AB_improved heuristic. This suggests that being in the centre of the board is not very important, while being in the corners is significant. **The added information and time complexity in look_ahead seems to be beneficial to the agent and allows for strong predictive power. This suggests that the added linear time complexity was worth the cost. For these reasons, I suggest the look_ahead heuristic in the real implementation.** In terms of the opponents, we see that AB_open does well because of its speed and simplicity. Ab_Center performs poorer that AB_open: this suggests that the centre is less important than intuition might suggest. The other MM opponents do not perform well because they do not use alpha beta pruning. The alpha beta pruning agents are able to search deeper and consequently can think more moves in advance.

Final Recommendation

In summary we recommend the look_ahead heuristic for these 5 reasons:

- Best winning rate
- It can look one move into the future for stronger predictive power and insight
- The heuristic makes sense intuitively
- The extra time complexity was indeed worth the cost
- A double jump is more telling of a descent move than that of a single jump