Heuristic analysis

Christopher Brian Currin

# Analysis

**def max\_player\_path**(game, player, max\_path=0.):  
 *"""  
 Near end game it is important to have depth-first over breadth-first search to determine winner  
 """* moves = game.get\_legal\_moves() # gets active players moves  
 **if** game.is\_winner(player):  
 **return** max\_path  
 **elif** game.is\_loser(player):  
 **return** 0  
 **for** move **in** moves:  
 new\_game = game.forecast\_move(move) # plays move into copy of the game (and changes active player)  
 mp = max\_player\_path(new\_game, player, max\_path + 1)  
 max\_path = max(max\_path, mp)  
 **return** max\_path

**def heuristic6**(game, player):  
 **return** max\_player\_path(game, player)

The heuristic searches until the end of the game and gives a score based on the maximum path it took. Future heuristics could find the minimum route to win.

As the heuristic essentially ignores the depth specified in minimax or alphabeta methods, this heuristic focuses on depth-first search rather than breadth-first search of possible outcomes. Because there are so many possibilities at the beginning of a game, this is very expensive, and is instead useful only towards the end of the game. Near the end of the game it is more useful to know who actually wins or loses than an estimate, and hence the usefulness of the heuristic.

**def max\_player\_path\_alt**(game, player, max\_path=0.):  
 moves = game.get\_legal\_moves() # gets active players moves  
 **if** len(moves) == 0:  
 **return** max\_path  
 **for** move **in** moves:  
 new\_game = game.forecast\_move(move) # plays move into copy of the game (and changes active player)  
 mp = max\_player\_path(new\_game, player, max\_path + 1)  
 max\_path = max(max\_path, mp)  
 **return** max\_path  
  
  
**def heuristic5**(game, player):  
 *"""  
 Returns the difference between the maximum depth a player can go for their moves  
 """* **return** max\_player\_path\_alt(game, player) - max\_player\_path\_alt(game, game.get\_opponent(player))

The heuristic Is similar to the previous one, except the value return from the subroutine (max\_player\_path\_alt) is a max\_path regardless of who wins or loses. The score returned by the heuristic is then a comparison between the player and the opponent.

**def heuristic4**(game, player):  
 *"""  
 The number of open spaces around the location of the player  
 """* location\_player = game.get\_player\_location(player)  
 blank\_spaces = game.get\_blank\_spaces()  
 open\_space\_player = 0.0  
 **for** (m, n) **in** [(1, 0), (0, 1), (-1, 0), (0, -1), (1, 1), (-1, -1), (1, -1), (-1, 1)]:  
 **if** (location\_player[0] + m, location\_player[1] + n) **in** blank\_spaces:  
 open\_space\_player += 1  
 **return** open\_space\_player

The heuristic values the amount of open space around a player. It does better than ID\_improved by a relatively small margin. Because it quantifies open space around a player and not actual moves, this can be misleading near the end of the game where moving to a space with open space around it may not actually be that useful for knight-style movement. It does however seem to perform quite well middle-game when mixed with other heuristics.

**def heuristic3**(game, player):  
 *"""  
 The number of open spaces around the location of the player's future move  
 """* best\_num\_open\_spaces = 0.0  
 **for** legal\_move **in** game.get\_legal\_moves():  
 new\_game = game.forecast\_move(legal\_move)  
 location\_player = new\_game.get\_player\_location(player)  
 blank\_spaces = new\_game.get\_blank\_spaces()  
 open\_space\_player = 0.0  
 **for** (m, n) **in** [(1, 0), (0, 1), (-1, 0), (0, -1), (1, 1), (-1, -1), (1, -1), (-1, 1)]:  
 **if** (location\_player[0] + m, location\_player[1] + n) **in** blank\_spaces:  
 open\_space\_player += 1  
 best\_num\_open\_spaces = max(best\_num\_open\_spaces, open\_space\_player)  
 **return** best\_num\_open\_spaces

Similar to the previous heuristic except it does an approximate lookahead for each possible move too (in an attempt to better the end-game poor performance), but does poorer in practice.

**def heuristic2**(game, player):  
 *"""  
 Difference between the number of open spaces around the location of the players  
 """* location\_player = game.get\_player\_location(player)  
 location\_opp = game.get\_player\_location(game.get\_opponent(player))  
 blank\_spaces = game.get\_blank\_spaces()  
 open\_space\_player = 0.0  
 open\_space\_opp = 0.0  
 **for** (m, n) **in** [(1, 0), (0, 1), (-1, 0), (0, -1), (1, 1), (-1, -1), (1, -1), (-1, 1)]:  
 **if** (location\_player[0] + m, location\_player[1] + n) **in** blank\_spaces:  
 open\_space\_player += 1  
 **if** (location\_opp[0] + m, location\_opp[1] + n) **in** blank\_spaces:  
 open\_space\_opp += 1  
 **return** open\_space\_player - open\_space\_opp

Similar to the previous heuristics (heurisic3 and heuristic4), but compares the score to the opponent, so that it favours moves that result in more open space than the opponent. In the early game this is therefore quite arbitrary, but definitely performs fairly well middle game. Performance is about the same as heuristic4 (slightly better than ID\_improved).

**def heuristic1**(game, player):  
 *"""  
 Number my moves vs opposition moves  
 similar to improved score  
 (no player win/lose values as this heuristic is used in the preferred heuristics where this is not necessary)  
 """* num\_moves\_player = len(game.get\_legal\_moves(player))  
 num\_moves\_opposition = len(game.get\_legal\_moves(game.get\_opponent(player)))  
 **return** float(num\_moves\_player - num\_moves\_opposition)

Similar to the improved\_score supplied heuristic, but included here due to slight performance improvement when mixed with other heuristics. That is, it does not check whether a move results in a win or lose; as in a mixed-heuristic method, this is typically done by a max\_path\_length heuristic such as heuristic 5 or 6.

Mixed heuristics

**def heuristic\_main**(game, player):  
 *"""  
 weight number of moves more highly at the beginning of the game, then amount of open space  
 and finally, maximum path length (winner will have max length)  
 each measure for player is compared to opponent's measure  
 """* prop = len(game.get\_blank\_spaces()) / total\_board\_spaces  
 **if** prop < 0.2:  
 **return** heuristic5(game, player)  
 **return** prop \* heuristic1(game, player) + heuristic2(game, player)

The heuristic calculates how far into the game the game it is (`prop`) and biases against heuristic1 over the course of the game. For end game (defined as prop < 0.2), heuristic5 is solely used, which tries to search to the end of the game for a winner. Performs consistently better than ID\_improved by about 10-15%.

**def heuristic\_main\_staggered**(game, player):  
 *"""  
 use number of moves at the beginning of the game, then amount of open space  
 and finally, maximum path length (winner will have max length)  
 each measure for player is compared to opponent's measure  
 """* game\_stage = len(game.get\_blank\_spaces()) / total\_board\_spaces  
 **if** game\_stage > 0.7:  
 # early game  
 **return** heuristic1(game, player)  
 **elif** game\_stage < 0.2:  
 # late game  
 **return** heuristic5(game, player)  
 **else**:  
 # middle game  
 **return** heuristic2(game, player)

Similar to previous heuristic, except heuristics are solely responsible for certain points in the game. The heuristic performs significantly worse than the above heuristic. Clearly a weighted balance between both heuristic1 and heuristic2 measures is better than a single measure, even if it is faster to computer a single heuristic.

**def heuristic\_mainalt**(game, player):  
 *"""  
 weight number of moves more highly at the beginning of the game, then amount of open space  
 and finally, maximum path length (winner will have max length)  
 """* prop = len(game.get\_blank\_spaces()) / total\_board\_spaces  
 **if** prop < 0.2:  
 **return** heuristic6(game, player)  
 **return** prop \* heuristic1(game, player) + heuristic2(game, player)

Similar to previous heuristic\_main, but uses heuristic6 instead of heuristic5 (see each one’s explanation for more information). Performs slightly better than heuristic\_main.

**def heuristic\_main\_alt\_01**(game, player):  
 *"""  
 weight number of moves more highly at the beginning of the game, then amount of open space  
 and finally, maximum path length (winner will have max length)  
 """* prop = len(game.get\_blank\_spaces()) / total\_board\_spaces  
 **if** prop < 0.1:  
 **return** heuristic6(game, player)  
 **return** prop \* heuristic1(game, player) + heuristic2(game, player)

**def heuristic\_main\_alt\_03**(game, player):  
 *"""  
 weight number of moves more highly at the beginning of the game, then amount of open space  
 and finally, maximum path length (winner will have max length)  
 """* prop = len(game.get\_blank\_spaces()) / total\_board\_spaces  
 **if** prop < 0.3:  
 **return** heuristic6(game, player)  
 **return** prop \* heuristic1(game, player) + heuristic2(game, player)

The above 2 heuristics are slight variations in terms of defining end game (prop).

**def heuristic\_main\_alt\_less**(game, player):  
 *"""  
 weight number of moves more highly at the beginning of the game, then amount of open space  
 """* prop = len(game.get\_blank\_spaces()) / total\_board\_spaces  
 **return** prop \* heuristic1(game, player) + heuristic2(game, player)

Only relies on heuristic1 and heuristic2.

**def heuristic\_main\_alt\_less\_more**(game, player):  
 *"""  
 weight number of moves more highly at the beginning of the game, then amount of open space  
 near end game (when improved score gives a winner or loser, prop won't have an effect due to the infinities)  
 """* prop = len(game.get\_blank\_spaces()) / total\_board\_spaces  
 **return** prop \* improved\_score(game, player) + heuristic2(game, player)

Similar to above, but uses provided improved score heuristic to the use of infinities for end game results, which should be favoured.

**def heuristic\_mainalt\_staggered**(game, player):  
 *"""  
 use number of moves at the beginning of the game, then amount of open space  
 and finally, maximum path length (winner will have max length)  
 each measure for player is compared to opponent's measure  
 """* game\_stage = len(game.get\_blank\_spaces()) / total\_board\_spaces  
 **if** game\_stage > 0.7:  
 # early game  
 **return** heuristic1(game, player)  
 **elif** game\_stage < 0.2:  
 # late game  
 **return** heuristic6(game, player)  
 **else**:  
 # middle game  
 **return** heuristic2(game, player)

Similar to heuristic\_mainalt and heuristic\_main\_staggered, and performs poorly.

# Performance table

|  |  |
| --- | --- |
| **Heuristic** | **Performance (% wins)** |
| ID\_Improved | 67.14% |
| heuristic\_mainalt | 73.57% |
| heuristic\_mainalt\_staggered | 62.86% |
| heuristic\_main\_staggered | 57.86% |
| heuristic\_main | 78.57% |
| heuristic\_main\_alt\_less | 72.86% |
| heuristic\_main\_alt\_less\_more | 73.57% |
| heuristic\_mainalt\_prop | 67.86% |
| heuristic1 | 57.86% |
| heursitic2 | 69.29% |
| heuristic3 | 57.86% |
| heuristic4 | 69.29% |
| heuristic5 | NA |
| heuristic6 | NA |

# Recommendations

heuristic\_main, heuristic\_mainalt and heuristic\_main\_alt\_less\_more have similar performance all consistently better than ID\_improved. My recommendation would be to heuristic\_main as it has the *best* (and *most consistent*) performance compared to ID\_improved. It also scales well in terms of computing power. In terms of scaling the size of the board, heuristic\_main\_alt\_less\_more may scale better due to avoid depth-first search after a relatively arbitrarily-defined end-game point.

# Possible improvements:

For long/complicated heuristics such as depth-first heuristics (heuristic5 and 6), it may be prudent to pass in the time\_left object to return *some* value in case of a time\_out (or close to it).

Consider reflections of the board near the beginning of the game to speed up computations so depth for iterative algorithms can be deeper.