Heuristic Analysis

Jason Mancuso

March 2017

Results

Table 1 shows a summary of my custom heuristics' performance relative to Improved_ID.

Table 1: Heuristic tournament results

Heuristic	**Improved_ID**	**Student**	**Relative Improvement**
Baseline	66.43%	76.29%	14.84%
Heuristic 1	67.00%	75.29%	12.37%
Heuristic 2	67.86%	72.43%	6.73%
Heuristic 3	69.29%	75.86%	9.48%
Heuristic 4	69.86%	77.00%	10.22%

Baseline

What I have termed "Baseline" is almost but not exactly identical to the Improved_ID heuristic. It is actually the same function that Improved_ID uses, only restricted to the range $[-8, 8]$, instead of $[-\infty, \infty]$. The only change made was to remove the checks for winning and losing matches. My implementation of alpha beta pruning already checks for winning states and doesn't need to check for losing states, so the custom_score function doesn't necessarily need to check. This is gives a somewhat biased measure of the Student, since Improved_ID therefore checks for a win/loss during both pruning and state evaluation. This is why I included Baseline as its own heuristic to compare mine against.

Results

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 77 to 23
  Match 2: ID_Improved vs   MM_Null     Result: 77 to 23
  Match 3: ID_Improved vs   MM_Open     Result: 64 to 36
  Match 4: ID_Improved vs MM_Improved   Result: 63 to 37
  Match 5: ID_Improved vs   AB_Null     Result: 67 to 33
  Match 6: ID_Improved vs   AB_Open     Result: 56 to 44
  Match 7: ID_Improved vs AB_Improved   Result: 61 to 39


Results:
----------
ID_Improved         66.43%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 99 to 1
  Match 2:   Student   vs   MM_Null     Result: 94 to 6
  Match 3:   Student   vs   MM_Open     Result: 70 to 30
  Match 4:   Student   vs MM_Improved   Result: 70 to 30
  Match 5:   Student   vs   AB_Null     Result: 75 to 25
  Match 6:   Student   vs   AB_Open     Result: 67 to 33
  Match 7:   Student   vs AB_Improved   Result: 59 to 41


Results:
----------
Student             76.29%
```

Code

```
def baseline_score(game, player):
    """Calculate the heuristic value of a game state
from the point of view
    of the given player.

    Taken from sample_players.py.

    Note: this function should be called from within a
Player instance as
    `self.score()` -- you should not need to call this
function directly.

    Parameters
    ----------
    game : `isolation.Board`
        An instance of `isolation.Board` encoding the
current state of the
        game (e.g., player locations and blocked cells).

    player : object
        A player instance in the current game (i.e., an
object corresponding to
        one of the player objects `game.__player_1__` or
`game.__player_2__`.)

    Returns
    -------
    float
        The heuristic value of the current game state to
the specified player.
    """
    return float(len(game.get_legal_moves(player))-
len(game.get_legal_moves(game.get_opponent(player))))
```

Heuristic 1

The idea for this heuristic was to weight the agent's potential moves more than the opponent's potential moves. The hope was that this would force the agent to focus on its own moves more than on its opponent's moves, resulting in a more defensive agent. The idea worked well. It performed best out of the custom heuristics analyzed.

Results

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 88 to 12
  Match 2: ID_Improved vs   MM_Null     Result: 67 to 33
  Match 3: ID_Improved vs   MM_Open     Result: 64 to 36
  Match 4: ID_Improved vs MM_Improved   Result: 58 to 42
  Match 5: ID_Improved vs   AB_Null     Result: 69 to 31
  Match 6: ID_Improved vs   AB_Open     Result: 67 to 33
  Match 7: ID_Improved vs AB_Improved   Result: 56 to 44


Results:
----------
ID_Improved         67.00%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 94 to 6
  Match 2:   Student   vs   MM_Null     Result: 83 to 17
  Match 3:   Student   vs   MM_Open     Result: 76 to 24
   Match 4:   Student   vs MM_Improved  Result: 71 to 29
  Match 5:   Student   vs   AB_Null     Result: 71 to 29
  Match 6:   Student   vs   AB_Open     Result: 72 to 28
  Match 7:   Student   vs AB_Improved   Result: 60 to 40


Results:
----------
Student             75.29%
```

Code

```
def custom_score1(game, player):
    """Calculate the heuristic value of a game state
from the point of view
    of the given player.

    Note: this function should be called from within a
Player instance as
    `self.score()` -- you should not need to call this
function directly.

    Parameters
    ----------
    game : `isolation.Board`
        An instance of `isolation.Board` encoding the
current state of the
        game (e.g., player locations and blocked cells).

    player : object
        A player instance in the current game (i.e., an
object corresponding to
        one of the player objects `game.__player_1__` or
`game.__player_2__`.)

    Returns
    -------
    float
        The heuristic value of the current game state to
the specified player.
    """
    return float(len(game.get_legal_moves(player))**2-
len(game.get_legal_moves(game.get_opponent(player))))
```

Heuristic 2

The idea for this heuristic was to implement the symmetric opposite of Heuristic 1, weighting the opponent's potential moves more than the agent's potential moves. The resulting agent would be more aggressive, but also more prone to stumble into poor game states. It performed worst out of the custom heuristics analyzed.

Results

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 80 to 20
  Match 2: ID_Improved vs   MM_Null     Result: 70 to 30
  Match 3: ID_Improved vs   MM_Open     Result: 65 to 35
  Match 4: ID_Improved vs MM_Improved   Result: 60 to 40
  Match 5: ID_Improved vs   AB_Null     Result: 73 to 27
  Match 6: ID_Improved vs   AB_Open     Result: 68 to 32
  Match 7: ID_Improved vs AB_Improved   Result: 59 to 41


Results:
----------
ID_Improved         67.86%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 93 to 7
  Match 2:   Student   vs   MM_Null     Result: 86 to 14
  Match 3:   Student   vs   MM_Open     Result: 69 to 31
  Match 4:   Student   vs MM_Improved   Result: 64 to 36
  Match 5:   Student   vs   AB_Null     Result: 75 to 25
  Match 6:   Student   vs   AB_Open     Result: 56 to 44
  Match 7:   Student   vs AB_Improved   Result: 64 to 36


Results:
----------
Student             72.43%
```

Code

```
def custom_score2(game, player):
    """Calculate the heuristic value of a game state
from the point of view
    of the given player.

    Note: this function should be called from within a
Player instance as
    `self.score()` -- you should not need to call this
function directly.

    Parameters
    ----------
    game : `isolation.Board`
        An instance of `isolation.Board` encoding the
current state of the
        game (e.g., player locations and blocked cells).

    player : object
        A player instance in the current game (i.e., an
object corresponding to
        one of the player objects `game.__player_1__` or
`game.__player_2__`.)

    Returns
    -------
    float
        The heuristic value of the current game state to
the specified player.
    """
    return float(len(game.get_legal_moves(player))-
len(game.get_legal_moves(game.get_opponent(player)))**2)
```

Heuristic 3

This heuristic was inspired by a project I worked on at school. I found that when tuning machine learning models, automatically tuned models performed better than many of my hand-tuned models. I decided to try to make the heuristic parametric and to optimize the parameters. The weights of the terms were found using Bayesian optimization. I performed this parameter search before I realized that compute time seems to be more significant than precision for these kinds of closed form heuristics. It seems that the time lost by computing the rather complicated closed form expression outweighs the accuracy gained for evaluation purposes.

Results

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 87 to 13
  Match 2: ID_Improved vs   MM_Null     Result: 78 to 22
  Match 3: ID_Improved vs   MM_Open     Result: 58 to 42
  Match 4: ID_Improved vs MM_Improved   Result: 70 to 30
  Match 5: ID_Improved vs   AB_Null     Result: 71 to 29
  Match 6: ID_Improved vs   AB_Open     Result: 61 to 39
  Match 7: ID_Improved vs AB_Improved   Result: 60 to 40


Results:
----------
ID_Improved         69.29%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 97 to 3
  Match 2:   Student   vs   MM_Null     Result: 90 to 10
  Match 3:   Student   vs   MM_Open     Result: 70 to 30
  Match 4:   Student   vs MM_Improved   Result: 71 to 29
  Match 5:   Student   vs   AB_Null     Result: 75 to 25
  Match 6:   Student   vs   AB_Open     Result: 66 to 34
  Match 7:   Student   vs AB_Improved   Result: 62 to 38


Results:
----------
Student             75.86%
```

Code

```
def custom_score3(game, player):
    """Calculate the heuristic value of a game state
from the point of view
    of the given player.
    Note: this function should be called from within a
Player instance as
    `self.score()` -- you should not need to call this
function directly.
    Parameters
    ----------
    game : `isolation.Board`
        An instance of `isolation.Board` encoding the
current state of the
        game (e.g., player locations and blocked cells).

    player : object
        A player instance in the current game (i.e., an
object corresponding to
        one of the player objects `game.__player_1__` or
`game.__player_2__`.)
    Returns
    -------
    float
        The heuristic value of the current game state to
the specified player.
    """
    player_moves = game.get_legal_moves(player)
    opp_moves =
game.get_legal_moves(game.get_opponent(player))
    x = len(player_moves)
    y = len(opp_moves)
    return
float(82.874576048706842*x**7.42059917759554 64+82.874576
048706842**x-73.420012687089994*y**7.6446303872568819-
5.7974792415172294**y)
```

Heuristic 4

The idea for this heuristic was to change the opponent's contribution to the value for states with low numbers of moves. The heuristic performed slightly worse than Heuristic 1, suggesting that the extra compute of multiplying a weight on the opponent's number of moves was not worth it.

Results

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

Playing Matches:
----------
  Match 1: ID_Improved vs   Random      Result: 84 to 16
  Match 2: ID_Improved vs   MM_Null     Result: 76 to 24
  Match 3: ID_Improved vs   MM_Open     Result: 73 to 27
  Match 4: ID_Improved vs MM_Improved   Result: 55 to 45
  Match 5: ID_Improved vs   AB_Null     Result: 75 to 25
  Match 6: ID_Improved vs   AB_Open     Result: 62 to 38
  Match 7: ID_Improved vs AB_Improved   Result: 64 to 36


Results:
----------
ID_Improved         69.86%

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

Playing Matches:
----------
  Match 1:   Student   vs   Random      Result: 96 to 4
  Match 2:   Student   vs   MM_Null     Result: 88 to 12
  Match 3:   Student   vs   MM_Open     Result: 72 to 28
  Match 4:   Student   vs MM_Improved   Result: 73 to 27
  Match 5:   Student   vs   AB_Null     Result: 80 to 20
  Match 6:   Student   vs   AB_Open     Result: 64 to 36
  Match 7:   Student   vs AB_Improved   Result: 66 to 34


Results:
----------
Student             77.00%
```

Code

```
def custom_score4(game, player):
    """Calculate the heuristic value of a game state
from the point of view
    of the given player.

    Note: this function should be called from within a
Player instance as
    `self.score()` -- you should not need to call this
function directly.

    Parameters
    ----------
    game : `isolation.Board`
        An instance of `isolation.Board` encoding the
current state of the
        game (e.g., player locations and blocked cells).

    player : object
        A player instance in the current game (i.e., an
object corresponding to
        one of the player objects `game.__player_1__` or
`game.__player_2__`.)

    Returns
    -------
    float
        The heuristic value of the current game state to
the specified player.
    """
    return float(len(game.get_legal_moves(player))**2-
2*len(game.get_legal_moves(game.get_opponent(player))))
```

Conclusion

The Baseline heuristic has a clear advantage in gameplay performance against the other heuristics investigated. It takes into account both player's options. It does so while limiting the evaluation time of the custom_score function. I recommend using the Baseline heuristic.