Isolation Heuristic Functions

Table of Contents

1 Introduction

This report describes the implementation and assessment of three additional heuristic functions for the Isolation board game on a 7x7 grid.

Four heuristic functions are implemented with the sample Isolation game-playing code provided with this assignment. The first three supplied functions are `null_score` (always returns 0), `open_move_score` (returns the number of moves available to the player) and `center_score` (returns the square of the distance from the centre of the board to the player's position), none of which will be discussed further.

The fourth supplied heuristic is called `improved_score`, and, in contrast with `open_move_score`, is the *difference* in the number of moves available to each player from their current position.

A heuristic scoring function is one thing, and the algorithm which uses it is another. The game-tree search algorithms used in this assignment are Minimax and Alpha-beta pruning, using iterative-deepening search (sometimes with no fixed depth limit, at other times down to depths of 3 or 5 game plies.)

Over and above these heuristics, three more custom scoring heuristics have been implemented, and their performance compared against the first three supplied heuristics. The fourth supplied heuristic, `improved_score`, is also compared against the first three supplied, and finally my custom heuristics is matched up against `improved_score`.

2 Custom Heuristics

2.1 Weighted Moves

The first custom heuristic is a modification of the `improved_score` heuristic, based on an idea from GitHub user `on2valhalla` (Jason Mann). It's the difference between the number of moves available to the current player, minus three times the moves available to the opponent. The constant of 3 was determined by Jason to give better results in the opening stages of a game – see https://github.com/on2valhalla/Isola.) This difference is multiplied by the number of filled squares on the board.

In other words, a player has to have markedly more moves available than the opponent to get a positive score, and small differences in the players' available moves take on greater significance as the game progresses and there are fewer open spots left on the board.

2.2 Reachable Squares

Instead of simply counting how many *moves* are open to each player, this heuristic counts how many *positions on the board* a player can potentially reach. This heuristic is intended to compensate for the horizon effect.

Consider the situation where a player is caught in a cul-de-sac, and only has one move available (which would yield a score of 1 under the `improved_score` heuristic.) Making that move takes the player to a larger, less-occupied area, where it has more room to manoeuvre. The additional moves available to the player at this new position - and all positions reachable from them - are reflected in this heuristic score.

This heuristic rewards players who can potentially cover, or reach, a larger part of the game board. It doesn't say anything about if, or how likely it is that, the player's opponent will cut off that player.

2.3 Cut-off Reachable

This heuristic is a similar to the previous `reachable_score` heuristic. However, instead of calculating the simple difference between the amount of squares a player can potentially reach, it is the difference between the number of squares that each player, *and each player alone*, can potentially reach (in other words, the disjunction of the sets of positions available to each player.)

The score is bumped up (by an amount equivalent to the size of the board) if there are no common squares both players can potentially reach - in other words, if one player has managed to cut off the other.

The idea behind this heuristic is to steer players to positions where they isolate their opponents to a part of the board where they have less space in which to move around than the attacking players. The attacker should be able to comfortably win.

2.4 Implementation Complexity

All three heuristics potentially examine every cell on the board. The `weighted_moves_score` heuristic counts blank cells (which, with the reference Board implementation given, means iterating through every cell on the board), and the `reachable` heuristics may examine every cell for just one player–so all three heuristics are at least O(width x height).

The `reachable` heuristics may then examine all board cells again when calculating a score for the second player. Where they differ is that the `cut_off_reachable_score` heuristic will then additionally do three set operations (again, all of a big-oh order linear in the size of the board) and calculate a few simple (constant-time) mathematical expressions.

Though not as simple as the `weighted_moves_score` heuristic, the `cut_off_reachable_score` heuristic is only slightly more computationally expensive (but on the same order of complexity) as the `reachable_score` heuristic.

3 Results

Three agents were set up, one agent for every one of my custom heuristics. All three agents always

used Alpha-beta pruning and iterative deepening (ID) search, with an unlimited search depth (well, unlimited with respect to the time constraint per turn...)

As a basis for comparison, an agent was also set up to use the supplied `improved_score` heuristic, also with Alpha-beta pruning and unlimited ID search – this agent will be referred to as the ID Improved agent.

These two sets of agent(s) were played off, variously against each other and against agents using the four supplied heuristics, as described next.

3.1 Comparison with Supplied Heuristics

Four agents, each one using one of the four supplied heuristics were set up. These agents used alternately the Minimax algorithm, searching to a depth of 3 plies, and the Alpha-beta pruning algorithm, searching to a depth of 5 plies.

One further agent was created, which returned a move chosen randomly from the set of legal moves available.

40 matches were played between all agents, with the following results. The yellow-highlighted cells indicate the highest scoring custom heuristic.

Minimax 3 plies	**Weighted Moves**		**Reachable**		**Cut-off Reachable**		**ID Improved**	
	Won	Lost	Won	Lost	Won	Lost	Won	Lost
Random	32	8	38	2	36	4	38	2
Null	34	6	36	4	33	7	38	2
Open	27	13	20	20	25	15	25	15
Improved	24	16	23	17	26	14	26	14
Center	32	8	32	8	34	6	33	7
Win Ratio	74.50%		74.50%		77.00%		80.00%	

Results from playing agents, using the supplied heuristics and the Minimax search algorithm to a depth of 3 plies, against my custom agents and the ID Improved agent. Note that the choice of algorithm is irrelevant for the Random agent; the results for this agent are arbitrarily included here.

Alpha-beta 5 plies	**Weighted Moves**		**Reachable**		**Cut-off Reachable**		**ID Improved**	
	Won	Lost	Won	Lost	Won	Lost	Won	Lost
Null	29	11	31	9	30	10	29	11
Open	28	12	22	18	28	12	29	11
Improved	27	13	17	23	16	24	24	16
Center	34	6	27	13	30	10	31	9
Win Ratio	73.75%		60.63%		65.00%		70.63%	

Results from playing agents, using the supplied heuristics and the Alpha-beta search algorithm to a depth of 5 plies, against my custom agents and the ID Improved agent.

3.2 Custom Heuristics versus ID Improved

In this round, agents using my custom heuristics were played off solely against the ID improved agent. In each match (consisting of 5 games) both players alternated in playing first from a random starting position, giving a total of 10 games per match. 10 matches were played in each run, and 3 runs were played to get the results shown here.

The following tables show the scores obtained by aggregating the first run (200 games), then the first two runs (400 games), and finally, all three runs (600 games.)

200 games	versus ID_Improved		
	Won	Lost	Advantage
Weighted Moves	99	101	49.50%
Reachable	102	98	51.00%
Cut-off Reachable	105	95	52.50%

400 games	versus ID_Improved		
	Won	Lost	Advantage
Weighted Moves	194	206	48.50%
Reachable	202	198	50.50%
Cut-off Reachable	190	210	47.50%

600 games	versus ID_Improved		
	Won	Lost	Advantage
Weighted Moves	301	299	50.17%
Reachable	291	309	48.50%
Cut-off Reachable	310	290	51.67%

Results in the table are highlighted when an agent using a custom heuristic wins more games than the ID Improved agent.

4 Observations & Conclusion

The custom agents had no problem holding their own against agents using the supplied heuristic, irrespective of which search algorithm the supplied agents used. The custom agents beat the supplied agents at least 60.63% of the time, and the best performance (77.00%) was attained by the `cut_off_reachable` agent. None of the custom agents were able to out-perform the ID Improved agent, however, which won 80% of its matches.

Competing directly against each other, it doesn't seem as if any of the custom heuristic agents have an advantage over the ID Improved agent–or vice versa. The biggest advantage is 2.5%, and is attained both by the `cut_off_reachable` agent in the 200 game aggregate (representing a 5-game lead), and also by the opposing ID Improved agent in the 400 games result set (a 10-game lead.)

An advantage is not absolute - that is, one heuristic does not always dominate the other. Up to the 400 match aggregate, the `reachable_score` heuristic always beats the `improved_score` heuristic, but not when played over 600 games. The `cut_off_reachable_score` wins over 200 games, loses over 400, and wins again over 600 games.

The results would seem to indicate that the custom agents and the ID Improved agent are evenly matched, with the result of their competitions being influenced more by the initial board conditions than any inherent strength.

If we'd have to chose among heuristics, however, the results seem to suggest going with the `cut_off_reachable` agent, for the following reasons:

1. Good all-round performance: this agent attains the highest score against minimax agents, the second highest score against alpha-beta agents, and manages to beat the ID Improved agent.
2. Greatest advantage: when the `cut_off_reachable` agent does manage to beat the ID Improved agent, it does so with the the greatest lead (1.67% and 2.50%) compared to agents using the other heuristics (but it also had one of the largest losses–2.50%.)

3. Implementation complexity: Although not as simple to implement as the `weighted_moves_score` heuristic, the additional coding required for the `cut_off_reachable_score` heuristic is warranted by the reasons given in points (1) and (2) above. When compared to the `reachable_score` heuristic, it is of the same order of complexity (see 2.4 Implementation Complexity), conceptually still simple to understand, and performs as well or better.

5 Discussion & Further Work

I have not examined how many plies each heuristic can search per turn, to see if one heuristic can search/see further than the others, although intuitively I'd expect the `reachable` heuristics to search down to the same number of plies, which would be the same or fewer plies as those examined by the `weighted_moves_score` heuristic.

Several other potential heuristics were suggested, but weren't implemented. These include

1. A heuristic which penalizes a player if he moves too close to a corner, depending upon the difference in valid moves between the player and the opponent), and
2. A heuristic which normalises the difference between the player's and the opponent's legal moves against the total legal moves.

To be convincingly and consistently successful, it would seem an AI would have to have a better heuristic, a better strategy and take into consideration how a game is played.

For instance, the tournament script allows a player only a fixed amount of time to evaluate moves. That sort of environment favours heuristics which are quick and simple to evaluate (like `improved_score`), which only considers available moves, as opposed to the others, which not only have to calculate available moves, but also all positions reachable from them... and all available moves from them. What if the tournament script allowed each player a fixed amount of *moves* it could evaluate? That would allow for more expensive heuristics, and shift the focus onto the search strategy, like using alpha-beta pruning instead of minimax to take full advantage of the allocated quota of moves.

Another facet to strategy is knowing when to apply which heuristic. For example, all the heuristics discussed here aren't likely to provide much guidance to a player (or differentiate between the two players) in the opening stages of a game, as they return scores that are largely equal. When a game is starting and the board is still relatively clear, each player will have roughly the same amount of moves available at each turn, and be able to reach the same number of squares. A better idea would be to use a quick, cheap heuristic (like `improved_score`) or even better, a pre-computed opening book, in the opening stages of the game, and then a heuristic like `cut_off_reachable` at later stages, when there is both a greater chance that one player can cut off another and that those opportunities are likely to be only a few plies ahead.