Heuristics for Isolation game agent

Oleg Medvedev

December, 2017

Problem statement

Isolation board game played on large enough board has a significant breadth and depth to make the complete search of the whole game tree unfeasible. The solution is to employ techniques like iterative deepening to explore the tree only up to given depth and employ heuristic evaluation functions to evaluate the current board state to reduce breadth. The quality of the evaluation function has a strong impact on the overall performance of the agent. So these functions have to correlate well with the winning chances of the current player and be easily computable at the same time.

Evaluation approach

Instructors provided a `tournament.py` script to assist with evaluation of different score functions. It runs a series of matches (10 by default) of two rounds to give each player the initiative. To eliminate the influence of the first moves and remove the possibility of using opening books the script chooses the first two moves within each match randomly. While this evaluation approach is correct in principle it is not entirely complete, because it can be easily shown that depending on the random seed the results may differ significantly and change the final conclusion entirely.

To illustrate the point I modified the `tournament.py` script to run `NUM_REPEATS` number of tournaments, storing the outcome of each tournament in array so I can perform the hypothesis testing. I've chosen all four provided sample agents: `Random`, `AB_Open`, `AB_Center`, `AB_Improved` and matched them against `AB_Improved`, each match consisted of 10 rounds (20 games total). Every match was repeated 20 times.

The results of all 20 matches are stored in `tournament_sample.log` file in the repository. I ran a two-sample hypothesis testing between results for each agent vs `AB_Improved` and results of `AB_Improved` vs itself as a benchmark. The idea is that if the performance of the agent vs `AB_Improved` is truly different it will be reflected both in the mean win/loss ratio and in p-value.

Results are shown below.

Agent	W/L mean	W/L std	p-value	95% significant
Random	0.130	0.116	0.0000	True
AB_Open	1.005	0.141	1.0000	False
AB_Center	0.890	0.171	0.0066	True
AB_Improved	1.005	0.148	1.0000	False

One may see that as expected `Random` agent performs very poorly, so there is really zero chance that it is similar to `AB_Improved` performance. `AB_Center` has a lower mean W/L ratio and p-value is well below 0.05 or even 0.01 significance limit. The non trivial result is that `AB_Open`, which is as per course materials is assumed to be inferior to `AB_Improved` has actually similar performance when you increase the sample size (20 tournaments of 20 matches each). So, while `AB_Improved` heuristics in principle should indeed provide an improvement over `AB_Open` it likely does so only in some cases.

Considered evaluation functions

One may derive a large number of possible evaluation functions based on the methods available in `Board` class. It is possible to use different combinations of the number of legal moves for the player and the opponent, calculate Manhattan or geometric distance between players or distance to center or edges of the board, combine different metrics, etc.

After experimenting with a large number of possible combinations I created the following short-list of evaluation functions.

1. Aggressive agent

This function is based on `AB_Improved` with two changes. First it introduces the coefficient which awards reducing opponent's legal moves. Second it awards getting closer to the opponent:

$$function = N_{player} - kN_{opponent} - \hat{d}_{players}$$

where $\hat{d}_{players}$ is the normalized Manhattan distance between players. The reason for normalization is to make sure that function still prefers actions with more available moves and just choose to move closer to opponent for otherwise similar moves.

Empirically I've chosen the value of k to be 1.6.

2. Defensive agent

Similar to the previous one in structure, however the logic is opposite:

$$function = kN_{player} - N_{opponent} + \hat{d}_{players}$$

This function rewards increasing own moves and rewards getting farther from the opponent. Similar to the previous agent the k value of 1.6 seems to be the optimal one.

3. Cautious agent

Finally this function also rewards increasing the number of own moves, but instead of rewarding getting farther from the opponent it rewards staying closer to the center:

$$function = kN_{player} - N_{opponent} - \hat{d}_{center}$$

Here the $\hat{d}_{center}$ is the normalized Manhattan distance to the center of the board.

Other functions

I considered two other functions as well. One was penalizing positions on the edge of the board with double penalty for position in any corner. Another was a "lookahead" version of `AB_Improved`. It counted all available moves for the player and the opponent not only for the current board state, but also for the next move.

Results

As discussed in one of the previous sections I ran 20 tournaments and performed two-sample hypothesis testing. The raw results are stored in `tournament_agents.log`. Results of the statistical testing are summarized in the table below:

Agent	$\mu_{W/L}$	$\sigma_{W/L}$	p-value
Aggressive	1.005	0.107	0.0803
Defensive	1.020	0.160	0.0758
Cautious	1.025	0.118	0.0356
Avoid corners	0.955	0.140	0.5429
Lookahead	0.960	0.120	0.4522

`Cautious` agent is the only one which passed the hypothesis test with 95% confidence level, so as per the assignment it is the only one out of 5 which *consistently* outperforms `AB_Improved`. It is worth noting that both `Aggressive` and `Defensive` agents have a good mean W/L ratio. The p-value is quite small, so if we lower our confidence limit to 90% we can claim that they consistently outperform `AB_Improved` as well.

Recommendation

I recommend the `Cautious` agent based on the following three considerations:

- Performance. As per the `Results` section this agent is the only consistently outperforming the benchmark `AB_Improved` agent.
- Simplicity of the calculation. This agent does not introduce any significant complexity vs `AB_Improved`, as the only difference is the calculation of the normalized Manhattan distance, which has `O(1)` complexity.
- Compared to some other considered agents, like `Lookahead` this agent does not introduce any additional iterations of the game tree.