Heuristic Analysis: Isolation

Lee Honan, March 2018

This document discusses the approach that I used in the project, focusing on heuristics

---

General Approach

I played a few pen-and-paper games and concluded that the 'chess knight' variant of Isolation used in this project seems to need a different approach to that of the 'sliding pieces' Isolation shown in the classes. It is difficult for competent players to block one-another from moving in the early stages of the game, and occupying the centre of the board early is not beneficial.

I came across the 'knight's tour' puzzle; where Warnsdorff's algorithm allows all squares to be visited by visiting the squares with the fewest available moves first on an even-sided symmetrical board larger than 4*4). For other board shapes the knight's tour approach still seems to be an adequate if not optimal strategy. The game can be seen as an adversarial knight's tour.

There is an obvious trade-off between using a simple heuristic(s) that execute quickly and allow for a greater number of nodes to be evaluated, and a more complex heuristic(s) that provides a more accurate assessment at the cost of search depth. Of course, more costly heuristics can be employed selectively, such as when assessing a smaller set of pre-qualified nodes (or ordering the nodes to be expanded given an earlier, faster heuristic analysis).

For the project I wanted to explore these trade-offs and experiment with a few supplemental techniques including caching evaluations (with the option of rotating and flipping the board state to find matches), and using an opening book (I later realised that this was prohibited by the tournament.py program).

I developed a generic, parametric scoring function that could be used to try different strategies, with different weights given to different heuristics. Progress is measured, allowing for different tactics at different stages. This resulted in the following parameters:

- Knight's tour weight, applied up to game progress p%.
- Centrality weight, applied from game progress p%.
- Player move evaluation, looking for available legal moves at level 1 (from a position t level 0), with the option of conduction a beam search to look for up to n nodes at depth d with early termination if found. If a deeper search is performed each node is given an exponential weigh according to its depth.
- Opponent move evaluation (as for player move evaluation, but an inverse score).
- Whether to use the evaluation dictionary/cache, and if so whether to apply transformations when searching for matching game evaluations. The cache accumulates across games.

I also pre-sorted moves in the alphabeta function to evaluate those with the highest number of legal moves first. I implemented a slightly faster function to get legal moves.

I implemented three custom score functions, according to the following strategies:

- custom_score: An aggressive player that tries to follow a knight's tour path in the early stages of the game, then prefers centrality.
- custom_score_2: A staged player that starts aggressively, becomes increasingly defensive, uses a shallow beam search of own or opponent's moves, and ignores alignment with knight's tour path or centrality.
- custom_score_3: A defensive player that tries to follow a knight's tour path in the early stages of the game, then prefers centrality.

Results

I ran tournament.py with 10 games instead of 5 per match to get a more reliable measurement of relative performance. The results are given below.

Match #	Opponent	AB_Improved		AB_Custom		AB_Custom_2		AB_Custom_3	
		Won	Lost	Won	Lost	Won	Lost	Won	Lost
1	Random	19	1	20	0	18	2	17	3
2	MM_Open	12	8	15	5	14	6	15	5
3	MM_Center	16	4	20	0	18	2	19	1
4	MM_Improved	13	7	17	3	16	4	15	5
5	AB_Open	9	11	9	11	11	9	5	15
6	AB_Center	13	7	12	8	9	11	17	3
7	AB_Improved	12	8	12	8	7	13	13	7
	Win Rate:	67.1%		75.0%		66.4%		72.1%	

Observations

1. The alignment with a knight's tour path seems of little value, as the piece is opposed. Following such a path (where cells that over the lowest move count are best visited first) appears unwise after ~20% the game.
2. Centrality seemed to offer some value in nudging the piece out of corners as the game progressed (especially if used with the knight's tour measure in earlier stages).
3. Pre-sorting moves in the alphabeta function to ensure those with the highest number of legal moves are evaluated first made a significant difference, improving win/loss rates by around 30%. The overhead of doing this appears more than offset by the benefit of exploring the most promising moves first.
4. The short move timeout favours simpler heuristics, and works against deep search extensions (which have an exponential cost with depth). Nonetheless using the beam search liberally seemed to help.
5. While it was difficult to isolate the difference that the cache made, it did not seem beneficial — assumedly because while hits average 10%, the majority are misses which add an overhead and thwart the ability to search deeper. Also the use of transformations to find matching evaluations was of little help as the board rapidly becomes asymmetrical with game progress.
6. A greater number of parameters and options creates a tuning problem, whereby a highly effective configuration may be possible but a great deal of work needs to be employed to find it;. For all but the most demanding applications making a simpler heuristic that is 'good enough' is a better choice.