Heuristic analysis

Christopher Brian Currin

Analysis

Heuristic6 (and with it max_player_path) 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.

Heuristic5 (and with it max_player_path_alt) 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.

Heuristic4's 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.

Heuristic3 is 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.

Heuristic2 is 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).

Heuristic1 is 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

Heuristic_main 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%.

Heuristic_main_staggered is 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.

Heuristic_mainalt is 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.

Heuristic_main_alt_01 and heuristic_main_alt_03 are slight variations in terms of defining end game (prop). Not much improvement (not enough rounds run to adequately determine best value).

Heuristic_main_alt_less is similar to heuristic_main but only relies on heuristic1 and heuristic2 (including until end game). Performs slightly better than either one by themselves.

Heuristic_main_alt_less_more is similar to above, but uses the provided improved_score instead of heuristic1 to make use of the infinities as possible return values for winning/losing; which is strongly favoured for end game results. Performs very well: similar to heuristic_main(alt).

heuristic_mainalt_staggered is 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 terms of the current systems it was tested on (decently-powered laptops), it is a good fit due to its relative simplicity early in the game that seems to create good positioning. Further, when winning/losing becomes more pertinent to making a decision, that is, the depth to reach states that determine winning/losing are more accessible in terms of searching on a regular computer, the heuristic switches to a more complex computation, but one which more accurately determines the outcome of the game. My recommendation is therefore also based on the heuristics balance between simplicity and complexity at different points in the game's progress.

A possible reason for its performance (well, for all the top 3 heuristics) is the considering of *both* players in determining a score. That is, a high score reflects the player does well in general, but

specifically in contrast to the opponent. Thus, each subcomponent (sub-heuristic) can also be weighted to more/less aggressively penalise the opponent and favour the player. Thus, the recommendation is also based on the tweak-ability of the heuristic to fit some situations, or fine-tune against an opponent over many trials.

Further motivation is that it also scales well in terms of computing power. In terms of scaling the size of the board, heuristic_main(alt) may need the end-game cut-off (the value of prop to 'switch' heuristics) to be altered. Another countering would be to use heuristic_main_alt_less_more, which may scale better as it is consistently a simple heuristic, but near winning/losing states gives infinities (plus/minus) as results; however, the search-depth of the alphabeta or minimax algorithm would need to be sufficient in searching deep enough.

In summary, the recommendation is based on:

1) It performs the best
2) It uses simple and complex sub-heuristics at 'appropriate' parts of the game state
3) It can be fine-tuned due to its multiple sub-heuristics (possibly towards specific 'types' of opponents)
4) It can scale fairly well with appropriate tuning for the given system and game (board) variation.

Essentially, the power of the heuristic comes in the form of using the 'right' heuristic at the right point in the game to choose a state that would be considered an advantage at *that* point in the game. The mix of heuristics is something I stumbled (hypotheses testing) upon, so the use of other heuristics in conjunction or instead of the ones used could provide further gains in performance.

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.