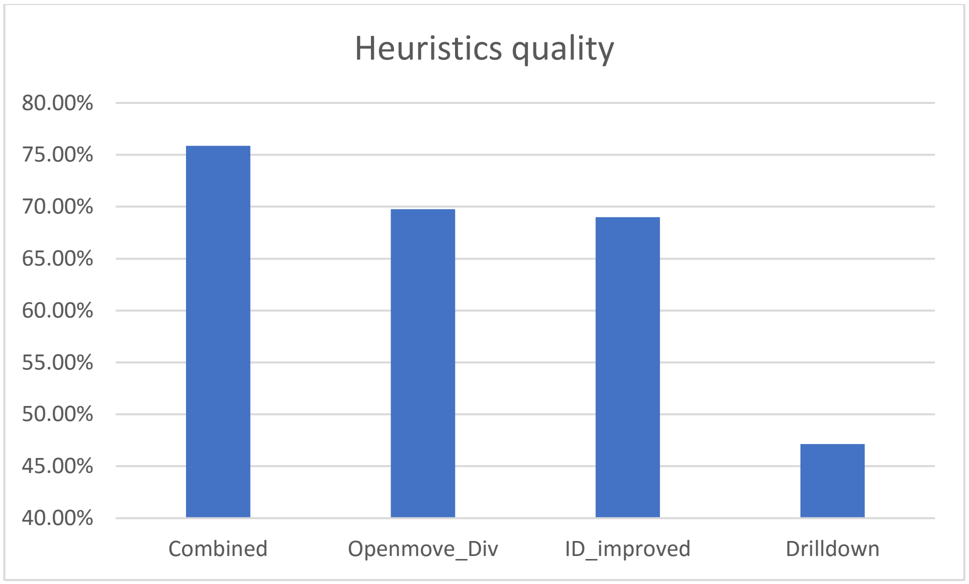
Summary

This is an analysis of heuristics implemented in AIND-Isolation project. It describes 3 out of 20+ heuristics considered as well as some generic conclusions.

Final test for all heuristic implementations (3 student's + ID_improved) included 5 tournaments runs for each. Average value of tournaments results is considered heuristic quality estimate.



The best result of 75.86% of wins is shown by combined heuristic using OpenMove_Div at the beginning of the game and Drilldown at the end.

OpenMove_Div heuristic

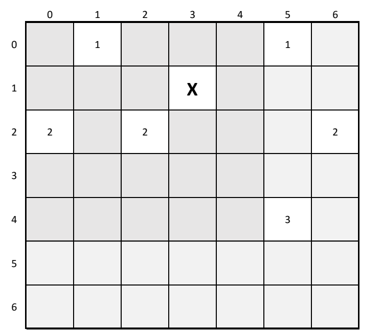
This heuristic is based on ID_improved provided in the course with two differences:

- Division is used instead of subtraction. The rationale behind in that division might reflect opponents' chances difference better in some situations. For example, 7 open moves vs 6 and 2 vs 1 returns same result (1) with subtraction. However, in practice first situation means almost equal chances while the second one might be close to win. Division result (1.17 and 2) reflects this difference better.
- Heuristic should account for active player. For many pre-terminal game states winner is defined by who's turn is next. Example of such position: only one blank field left, both players can move to that field. Thus, the following logic has been included into heuristic function: if player is not active and at least one of player's open moves is among opponent's open moves, one move is subtracted from player's score as it probably will be "closed" by opponent on next ply.

Heuristic score in the final test is 69.77%, which is only 0.77% better then original ID_improved score.

Drilldown heuristic

Any heuristic based on the number of open moves has one serious weakness: it is based on the "width" of available moves while game goal is to increase "depth" aka number of turns. Thus, this heuristic may lead to wrong decisions in situations like this:



Using open-moves-based heuristic, the player in position (1,3) will chose move to (0,1) as it provides two open moves. However, position (0,5) provides better "depth" and, thus, is better. In other worlds, open-moves-based heuristics are limited by analyzing single position only and don't look ahead.

An obvious way to improve is to implement heuristic that will analyze position several plies deeper[1]. The simplest and most obvious heuristic of such kind would be "maximum depth available" score. It was implemented and tested, but achieved comparatively low results.

Next, the following recursive heuristics called "drilldown" was implemented:

$$v(l) = \sum_{m \in M(l)} [r(m) * (d(m) + v(m))]$$

, where

l – current player location

$v(l)$ – location "value"

$M(l)$ – set of all blank board squares reachable from location l in one ply.

$d(m)$ – location "depth", number of moves needed to reach m from original player's location. Equals to 1 for directly reachable squares, to 2 at the second level etc. Is used to reflect "depth" value of the position.

$r(m)$ - passive player reduction coefficient less or equal to one.

Ideas behind this formula are:

- Analyze position several plies ahead. In contrast to minimax, however, this analysis do not include opponent moves and changes in available squares caused by those moves (with some correction for the first level of recursion).
- Account for maximum reachable depth by applying recursion and making deeper locations cost more by increasing $d(m)$ with each new level. Note: maximum recursion depth is regulated by parameter and is 5 in final implementation. Deeper recursion caused timeouts in tests.
- Account to number of available moves as well. Values of all locations at each recursion levels are summarized.

[1] An interesting argument against this approach would be "Stop, but we already have minimax implementation responsible for in-depth analysis! Heuristic should be limited to the single game state". However, an example of "Mastering the game of Go" research paper shows that in-depth heuristic might be useful. AlphaGo uses so called "fast rollout policy" to simulate game from the current node to the terminal state and then uses the result in node score calculation.
The obvious requirement to such in-depth heuristics is they should require less calculations then minimax implementation for the same depth.

- Account for the difference between active and passive player. Passive player reduction coefficient $r(m)$ is used if player is not an active and location m can be occupied by opponent at the next ply. This reduction is applied at the first level of recursion only.

For the heuristic calculation described value of player location is divided by value of opponent location.

A questionable part of this heuristic implementation is that it requires some “internal” knowledge of “isolation.py” implementation. Namely, `directions` list from `Board.__get_moves__` has been copied to heuristic code to allow drilling down without actually changing game state. Still, I don’t consider this a big issue. It can be resolved by implementing additional `Board` class method obtaining any board location and returning neighbor locations for it without checking if they are already occupied or not.

Against expectations, this heuristic demonstrated poor performance with just 47.14% quality. Presumably, the reason for this is that it almost “see” no difference between positions at the beginning of the game, acting only slightly better than random player. Another reason can be maximum minimax depth limitation at the beginning of the game caused by time limit and comparatively complex calculations in the heuristic function.

Despite poor performance, this heuristic in included into final version as combining it with OpenMove_Div demonstrates better results than OpenMove_Div itself.

Combined heuristic

Contradiction between expectations and real performance of Drilldown heuristic lead to the assumption that it is not suitable for game start, but can be used on a later stage. Closer to the game end branching factor reduces and allows deeper analysis even with comparatively complex heuristic. Also, at the game end length of the available path become much more important than number of open moves (because of, among other reasons, possible board partitioning).

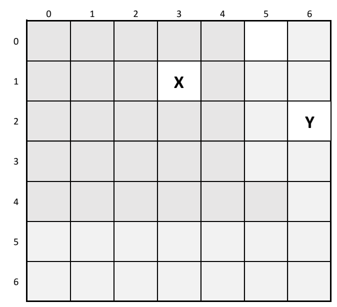
So, the next implemented heuristic is a combination of previous two:

- Free board part is calculated as number of blank spaces divided by the board size.
- If free board part is greater than 60% OpenMoves_Div heuristic is used. The 60% constant is a result of experiments.
- Otherwise, Drilldown heuristic is used.

This improvement allowed algorithm to achieve better results at the final stages of the game and reach 75.87% of wins (about 6% better than single OpenMoves_Div heuristic)

Lessons learned

- Manual analysis of game positions helps a lot in finding reasonable heuristics. Game playing helps even more.
- When there’s a need to compare the score of game opponents, two basic approaches are available: subtract one score from another or divide on score to another. In this task division works better. Both approaches should be considered at the very beginning of heuristic development.
- Each heuristic should manage to identify terminal states (wins and losses) and return appropriate score.
- Heuristic in adversarial games should account for active player. For many pre-terminal positions win or loss depends on who’s turn is next. For example:



- With time limit in place, there is a tradeoff between heuristic complexity (calculation time) and game tree search depth.
- Different heuristics might work better in different game states. Thus, it is useful to combine them to get the best of each.
- When several heuristics are combined for usage at different game stages, another tradeoff exists between heuristics analysis “depth” and the earliest game stage when this heuristic may be applied. The deeper analysis is the later it can be used. Time limit and high branching factor at the early game stages combined with “deep” heuristic lead to poor game tree depth search.
- Heuristics behavior testing and analysis require a lot of time even for such a sample game as 7x7 isolation. For more complex tasks it worth parallelizing this task to meet deadline.
- Report writing takes much more time than it is expected.