Artificial Intelligence Nanodegree

Adversarial search

Overview

The goal of this project is to create an adversarial search agent that plays the game “Isolation”. Each agent has a limited time to decide which action to take. The project focuses on **optimizing** the **Minimax** algorithm, **developing and comparing different heuristics**.

Minimax algorithm improvements

Minimax is a search algorithm that tries to minimize the possible loss in the worst case scenario. It builds a tree of scenarios by alternatively checking for the maximum outcome during its turn, and minimize the loss during its opponent turn.

When Minimax reaches the end of a game (win or loss) it propagates back this score up the tree of scenarios. Prior to reaching the end of a game different heuristics can be applied to estimate the current score of a tree node. These heuristics become important in a time constrained environment, as the Minimax algorithm may not have enough computational time to score to reach a deterministic solution.

To help improving the performance of the Minimax algorithm there are multiple algorithm enhancements that can be applied to the Minimax algorithm.

Alpha-beta pruning introduces a lower and higher limit on the score to help prioritizes the exploration of optimal nodes from the stand point of each player.

Iterative deepening forces Minimax to behave more like a BFS agent by putting a limit on the depth Minimax can reach at a given iteration. Once an iteration is complete, the depth limit is increased. It can be shown analytically that the cost of running the algorithm over multiple depth is negligible as each depth level double the time complexity. Limiting the depth can help reaching a better intermediary solution in a time constrained environment.

Custom Heuristics

- Heuristic 1: score = agent’s move – opponent’s move
  Aggressive strategies: minimize the opponent’s move
- Heuristic 2: score = agent’s move – **2x** opponent’s move
- Heuristic 3: score = agent’s move – **3x** opponent’s move
- Heuristic 4: score = agent’s move^2 – opponent’s move^2
- Heuristic 5: score = agent’s move^2 – **1.5x** opponent’s move^2
- Heuristic 6: score = agent’s move^2 – **2x** opponent’s move^2

Then from empirical analysis I observed that depending on the time given per turn certain heuristic performed better than others. This helped me realize that depending on how close the turn was to the end the heuristic would fair differently.
Based on this observation I introduced a logic to change the heuristic based on how many rounds have been played. Additionally from my experiments' observations I noticed that the average number of rounds to reach the game's end is around 33 rounds.
I introduce a round target which materializes when the agent should switch strategy. Using a grid search approach I determined that the locally optimal value is to switch strategy sometime after 10-15 rounds. I introduce a ratio of the current round / target round and switch strategy when the ratio passes 0.5.
Defensive to Aggressive strategies

- Heuristic 7: ratio < 0.5 uses inverse of heuristic 2, otherwise uses heuristic 2
- Heuristic 8: ratio < 0.5 uses inverse of heuristic 3, otherwise uses heuristic 3
- Heuristic 9: ratio < 0.5 uses inverse of heuristic 3, otherwise uses heuristic 5

Aggressive to defensive strategies

- Heuristic 10: ratio < 0.5 uses heuristic 2, otherwise uses inverse of heuristic 2 (#inverse of 7)
- Heuristic 11: ratio < 0.5 uses heuristic 5, otherwise inverse of heuristic 3 (inverse of #9)

Experiment

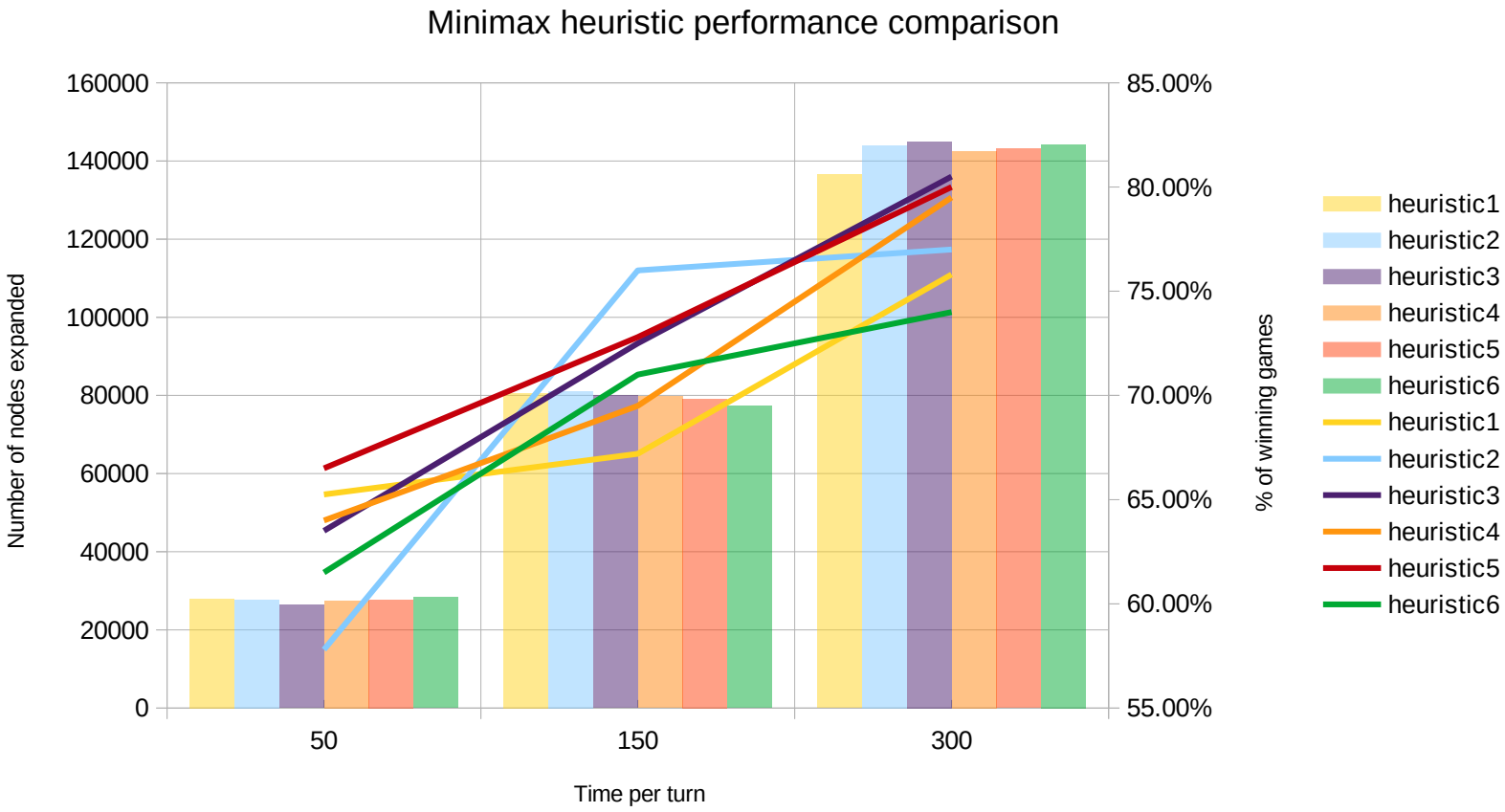
The baseline of the experiment is set as follow. The reference agent is a Minimax using heuristic 1, the custom agent is an Iterative deepening alpha-beta pruning Minimax which will be using all 10 heuristics to compare their performance. Additionally multiple time limits with be compared to assess the speed vs accuracy trade off.

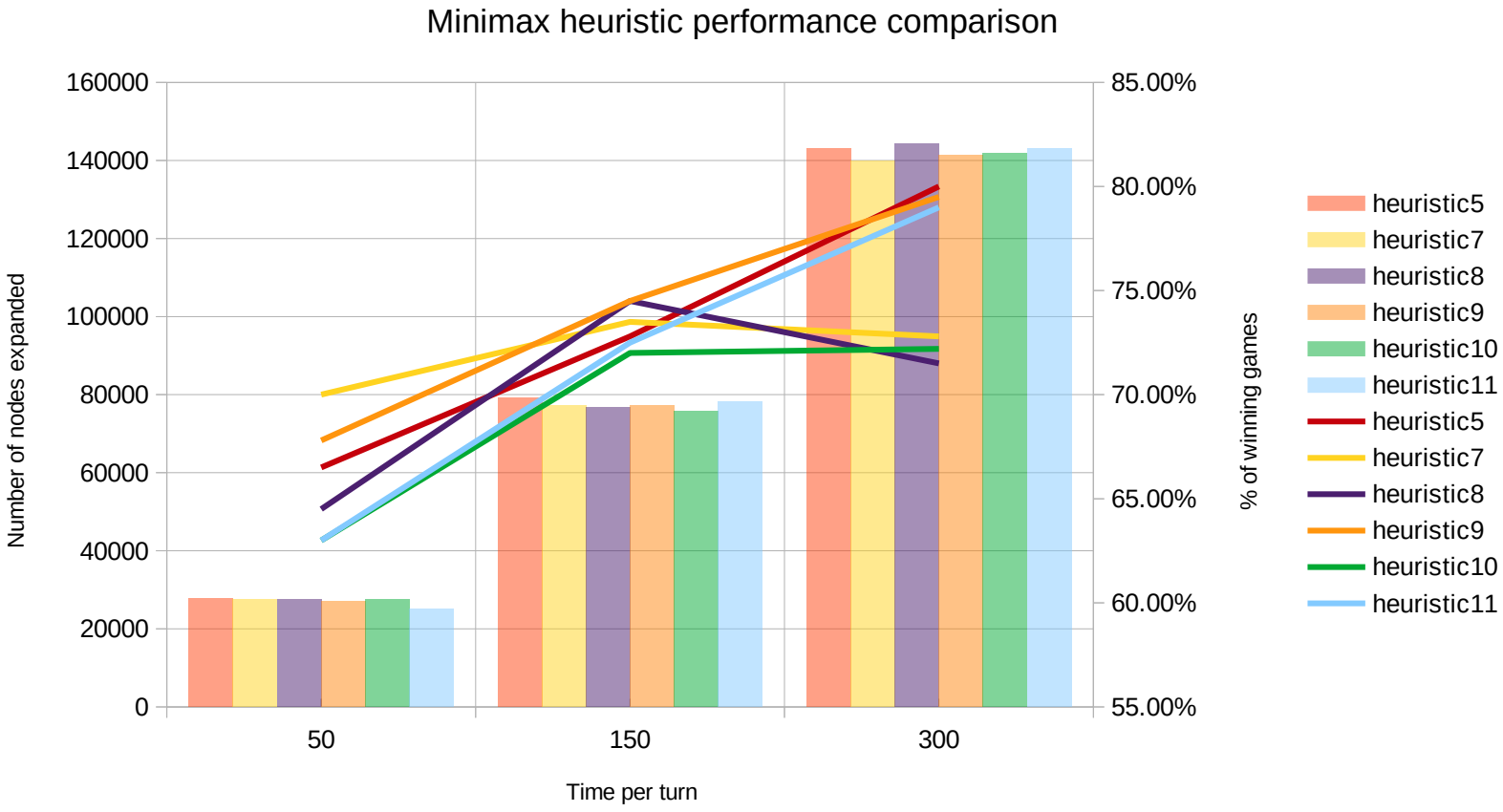
The custom agent is run 100 times with the fair_matches flag on, this results in 200 matches and should provide enough data to limit the randomness of results.

I have added one argument -u/heuristic to the command line which selects the heuristic to run on the custom agent.

```
python run_match.py -f -o MINIMAX -r 100 -u 8 -t 150
```

Results


Minimax heuristic performance comparison
160000
140000
120000
100000
80000
60000
40000
20000
0
85.00%
80.00%
75.00%
70.00%
65.00%
60.00%
55.00%
50
150
300
Number of nodes expanded
% of winning games
Time per turn
heuristic1
heuristic2
heuristic3
heuristic4
heuristic5
heuristic6
heuristic1
heuristic2
heuristic3
heuristic4
heuristic5
heuristic6


Minimax heuristic performance comparison
160000
140000
120000
100000
80000
60000
40000
20000
0
85.00%
80.00%
75.00%
70.00%
65.00%
60.00%
55.00%
50
150
300
Number of nodes expanded
% of winning games
Time per turn
heuristic5
heuristic7
heuristic8
heuristic9
heuristic10
heuristic11
heuristic5
heuristic7
heuristic8
heuristic9
heuristic10
heuristic11

Results evaluation

One can clearly see that both time per round and the choice of heuristic has an impact on performance. Certain heuristics also seem to be more stable with time.

The overall best heuristic is heuristic 5, which seems to be perform the best irregardless of the time per turn.

Questions

What features of the game does your heuristic incorporate, and why do you think those features matter in evaluating states during search?

The heuristics I developed introduces the number of rounds played as a feature. This feature has a clear impact on performance. It requires however a lot of time to find the locally optimal parameters (i.e. target rounds as well as aggressiveness and defensiveness factors). I used grid search to find a locally optimal target number of rounds.

Analyze the search depth your agent achieves using your custom heuristic. Does search speed matter more or less than accuracy to the performance of your heuristic?

I did not implement a variable to keep track of the depth reach by the algorithm but I will rather use the average number of nodes and rounds per game to estimate the average depth reach during each round. Let’s first take the upper limit of nodes 2^n for a given depth n, a rough average of 80,000 nodes per game and 33 rounds per game. This means an average of 2400 nodes per round, with 2^11 = 2048 that means the iterative deepening algorithms explores a minimum depth of 10 in its last iteration.

To simulate the speed vs accuracy trade-off, taking a look at the accuracy profile of different heuristics across multiple time-limits helps us understand which heuristics are most susceptible to a reduced speed.

Conversely the accuracy represents how certain the heuristic is of one move being higher value than another. Due to the nature of the iterative alpha-beta pruning, this means that unless the algorithm has reached all final possible states the accuracy can be assessed by the quality of the score at a non-final state.

An example of such trade-off is heuristic 2 and heuristic 7 which suffer degraded performance when allocated more time.