Heuristics Analysis

Introduction

The purpose of this project is to create an agent that will play the adversarial game, “Isolation”, incorporating the minimax algorithm for determining moves. In order to make these determinations efficiently within time constraints, alpha-beta pruning and iterative deepening search algorithms are also implemented (Russell & Norvig, 2010). The agent’s ability to beat opponents that include the same algorithms depends on the heuristics chosen in evaluating each minimax search level. In addition to the provided test heuristics, this project includes four additional heuristics. These were analyzed and tested so that a final recommendation could be used in the Student’s agent.

The project submission provided as follows based on the rubric:

- Game-tree searching: minimax and alphabeta methods
    - provided in code game_agent.py
- Agent success: CustomAgent class final performance
    - provided in code game_agent.py
- Evaluation function performance: results and analysis discussion
    - this paper: P1_heuristicsanalysis.pdf
- AI game-playing research: developments in the field of AI game-playing
    - provided in a separate report: P1_researchreport.pdf

1. Game-tree searching

The minimax and alphabeta methods pass

I successfully implemented minimax, alphabeta, and iterative deepening. All tests pass.

2. Agent success

The CustomAgent class beats the testing agents in a round-robin tournament when each player is limited to 150 ms per move. Winning agents perform better than ID_Improved against the test agents.

The final agent and custom_score() function produced varying results, presumably because it is very sensitive to the time constraint on my computer. The agent generally beats all test agents and outperforms ID_Improved.

3. Evaluation function performance

At least three evaluation functions other than the examples provided in sample_players.py are implemented and analyzed.

Four evaluation functions were implemented in addition to the null_score(), open_move_score(), and improved_score() functions provided. They are described here:

boxed_ratio_score() – calculates the ratio of legal_moves to valid_moves, where a “valid” move is one that is on the board but may or may not already be used. The purpose of this heuristic is to score the board position at any given time based on how “boxed in” the player is. For example, there may be fewer moves toward an edge, but if they are all available that might indicate that the area is relatively unexplored and might be a good location. The value lies between 0 and 1 to represent the percentage of open spots given the normal possible moves if the board was empty.

boxed_improved_score() – although this turned out to be a misnomer (it was not really an improvement), this is the difference between the player’s boxed_ratio and the opponent’s boxed_ratio, indicating the relative strengths of their “boxed” positions.

runaway_score() – this was meant as a purely defensive strategy. The value returned is the Manhattan distance between the player and opponent agents.

centerbias_score() – Experimentation showed that the improved_score() test heuristic was hard to beat. In an attempt to give that very good heuristic an edge, centerbias_score uses the Improved value and adds a bonus for center and off-center positions. These positions should already be favored by the Improved, but all things being equal, the center is still the most dominant position on the board. This heuristic boosts that so that the center and one off of center will be favored positions in the event of a tie.

In order to determine a ranking of these heuristics, each was run against the sample agents in tournament.py. The agents that employed alphabeta pruning and iterative deepening performed consistently better than those that did not use the iterative deepening algorithm, regardless of heuristic. Therefore, as a final faceoff, all of the Iterative Deepening (ID) agents were run against the other ID agents to determine which the best were and to normalize the rankings. The rankings, below, reflect these head-to-head competitions.

Agent	Win Rate
ID_Null (eval provided)	62.86%
ID_RunAway	62.86%
ID_BoxedImp	61.43%
ID_BoxedRatio	62.86%
ID_Open (eval provided)	52.86%
ID_Improved (eval provided)	67.86%
ID_CenterBias	69.29%

The runaway_score algorithm, predictably, performed about the same as Null, since it was purely defensive. Constant retreat may evade a random agent long enough for it to implode, but against an agent with a decent heuristic, runaway_score eventually boxes itself in a corner.

The boxed_ratio_score agent did pretty well (better than the Open agent), though overall they were evenly matched. This makes sense because the boxed_ratio_score gives a snapshot value of the board as a whole, taking into account the blocking positions already in place. Unfortunately, the boxed_improved_score heuristic was not an improvement, and performed poorly compared to everything but open_score, though sometimes beat boxed_ratio_score. The additional information about the opponent's blocked ratio was not helpful to the agent, and including it in the heuristic seemed to introduce randomness, reducing the usefulness of the original information.

Finally, the centerbias_score heuristic appears to be similar to the Improved heuristic, though the results were not consistent. The randomness of the starting positions used in the tournament, along with possible interference from the host computer during the limited time choices during iterative deepening, may be the cause of this uncertainty.

The top two agents, centerbias_score and Improved, both with Iterative Deepening, were run 120 times against each other to determine a final winner. The result was a tie. Either could be recommended for the game agent, but the centerbias_score heuristic was included in custom_score for the final submitted agent. The three reasons for the final agent recommendation are:

1. First and foremost both the improved_score and centerbias_score functions beat all other evaluation functions in head to head competitions.
2. In addition to the actual quantitative evidence supporting the choice of improved_score or centerbias_score, qualitatively we can think of what kind of information this evaluation function provides compared to the others. Null or Random do not evaluate the state of the board at all. Open only evaluates the state of the player. Since this is a two-player game, we really need to know the state of the board relative to the opponent's state. Improved does this by returning the difference between the player's open moves and the opponent's open moves, and therefore incorporates more information in its score. boxed_ratio_score also provided a relative score, but it is more derivative in nature takes longer to compute so not as good a choice.
3. In choosing between improved_score and centerbias_score, the data provided by the tournament.py program does not favor either. On one hand, we could argue that the simpler should be used (Improved). However, the tournament did not give the agents a choice of first move, whether either the first or second player. If the agents were given that choice, then their first move would favor the center spot over the near center spot when using centerbias_score, whereas Improved would see the center spot as "just as good" as the off center spots in a 7x7 game. Biasing towards the center position has the effect of providing an "opening book" via the evaluation function. This was a recommended strategy in the lectures,

so it is favored here, even though the tournament did not highlight this feature.

References

Russell, S., & Norvig, P. (2010). Artificial Intelligence: a modern approach.