Build a Game Playing Agent

Report by Sergio Lira

8/01/2018

Section 1 - Experiments with search techniques

For my project I chose to experiment with different search techniques. I started by implementing the minmax and alpha-beta search algorithms in my CustomPlayer to build a baseline. Then, implemented Principal Variation Search and used my selected baseline to compare its performance against minmax search.

Baseline experimentation Results

MinMax *depth = 3* (`python run_match.py -f -r 50 -p 4`)

- 50% win-rate: As expected.
- Using fair matches flag to ensure that the first player does not have the first-move advantage since none of the agents are leveraging opening moves. This flag extends to my others tests since none of my techniques are leveraging opening moves.

AlphaBeta *depth=3,4* (`python run_match.py -f -r 50, 100 -p 4`)

- Depth 3 -> 50% win-rate: As expected, useful to validate my implementation.
- Depth 4 -> 52.8% win-rate: Marginal gains.
- Depth 5 -> 60.0% win-rate.

AlphaBeta with Iterative deepening (`python run_match.py -f -r 100 -p 4 -t 150 ...`)

- Depth 5, timeout 150 -> 64.0% win-rate
- Depth 5, timeout 250 -> 68.0% win-rate

As expected AlphaBeta search provides performance gains proportional to the depth and timeout window values. As these two increase the speed advantage of AlphaBeta starts to show and wins more games when playing against a simple MinMax search. Specially if I leverage iterative search. I'll use AlphaBeta with iterative deepening as my baseline and test my search technique with the same hyperparameters of depth of 5, 100 rounds, timeouts of 150 and 250 milliseconds.

Section 2 – Principal Variation Search exploration

Principal Variation Search (PVS) like alpha-beta pruning intends to speed the search and computation of the minmax value of a node in a tree. The intuition is that alpha-beta search will work best in the scenarios were the first node searched is the best one, if we assume that this is the case we can search other nodes more quickly by using a null window, were the alpha and beta values are equal. If the assumption fails, the algorithm resorts to normal alpha-beta search. My implementation of PVS is based on the pseudo code found at: https://www.ics.uci.edu/~eppstein/180a/990202b.html

PVC experimentation Results

PVC with Iterative deepening (`python run_match.py -f -r 100 -p 4 -t 150, 250`)

- Depth 5, Timeout 150 -> 63.0% win-rate
- Depth 5, timeout 250 -> 71.5% win-rate

Section 3 – Advanced Search Techniques questions

Choose a baseline search algorithm for comparison (for example, alpha-beta search with iterative deepening, etc.). How much performance difference does your agent show compared to the baseline?

I observed almost equal performance with my implementation of PVC between. In the multiple runs tested I saw deltas of -/+ 2-3% points. I am unsure if I have a bug in my code. The pseudo code, implementations and papers I find online on PVC are meant to be used in chess games or have a substantially different implementation of alpha-beta.

Why do you think the technique you chose was more (or less) effective than the baseline?

According to the literature PVS should in most cases and with enough testing be more effective than alpha-beta pruning with iterative search. I may have underestimated the reliance of PVC in selecting a good first move to speed consecutive searches. In leu of a hashed table of good moves the performance of PVS may have degraded to the level of normal iterative alpha-beta.