Research Review: Game Tree Searching by Min / Max Approximation*

Loay Ashraf

New techniques introduced

The paper introduces a new technique for searching game trees in an efficient manner, briefly the idea is to approximate the min and max operators using generalized mean-value operators.

This is done by implementing new algorithm in C and testing it against iterative deepening with alpha-beta pruning algorithm in connect four game.

In detail the algorithm consists of the following:

1. Static evaluator function
2. Penalty based iterative searcher

The algorithm works by combining the two basic components to determine the next tip to expand which has the least penalty score possible, also the tip which has the biggest effect on the total score of the root node *s*, this root sensitivity can be measured by the partial derivative of score value of the root node *s* with respect to score value of any node *x* in the tree and this is done using the chain rule.

For example, if we have a tree C with root node *s* and the tree was expanded down to node *c* at depth *d* now we have to choose one node from the children nodes of *c* to expand, according to the algorithm we will have to draw several paths from the root node *s* to each of the children nodes of node *c* and then summing nodes weights for each path drawn and finally deciding on which node to choose based on penalty scores of each individual path drawn from the root node *s*.

Paper Results

As mentioned above the new algorithm was put to test against iterative deepening with alpha-beta pruning algorithm in connect four game, about 980 games were played with 49 different starting positions for each player, 490 time-bound games, 490 move count-bound games.

Then initial results indicated that the AB with pruning algorithm was far superior in time-bound games by winning 239 games versus 186 games won by MM algorithm but if we look at move

count-bound games the story is reversed with MM algorithm winning 249 games versus 190 games won by AB algorithm.

As mentioned in the paper the time-bound results could be improved by reducing the computational overhead per call to move operator, also some of the drawbacks of using this algorithm is mentioned:

1. it requires more memory resources as the tree being explored is required to be explicitly stored.
2. it's oriented towards improving the score at the root where possible instead of selecting the best move to take from the root, for example, if there is exactly one move to make from the root, then the algorithm will search the subtree under this move even though such exploration won't affect the decision made at the root.
3. it spends a lot of time traversing back and forth between the root and the leaves of the tree which in turn consumes a lot form the CPU time.