RESEARCH REVIEW

Game Tree Searching by Min-Max Approximation

Ron Rivest

Summary

Game playing by a computer is usually achieved by creating game trees where each node represents a particular state in the game. The game tree starts out from the initial configuration or state, also known as the root, and keeps branching out, exploring different possibilities based on the legal moves in continuation of the current node. When branching of a node is not possible, the node is said to be a terminal node. Assuming a two player game with perfect information, min-max is the technique where terminal nodes are assigned scores based on the favorability of the state for the player in question, and back propagating this value, applying minimization and maximization at alternate levels of the tree. This makes optimal play easy as the agent needs to follow that path that promises the maximum value for the terminal node.

Exploring the entire game tree and computing the actual values for the terminal nodes is easy for simple games with small game trees. However, for complicated games, the tree becomes huge and computing the actual value is impossible. In such cases, the game trees are searched till only a limited depth and an evaluation function is used as a heuristic to compute a proxy value for non-terminal node states, which are then back-propagated. Techniques such as alpha beta pruning and iterative deepening further increase the efficiency of searching the game tree.

The heuristic used above could be static or iterative. Which using static heuristics, the tree is expanded uniformly in terms of the depth. However, for iterative heuristics, the tree is expanded non-uniformly, with some nodes being expanded to a greater depth than others.

This paper introduces an iterative heuristic such as this, for game playing, based on min-max optimization. In using iterative heuristics, the choosing of an expandable tip of the game tree is crucial, while the rest of the process works similar to that while using a static one. The paper introduces an approach based on penalty-based schemes. In penalty based schemes, while exploring the tree, weights are assigned to every edge between a parent and child nodes. If a particular node is considered for expansion, the penalty for expanding that node is the sum of all the penalties between the edges from that node upto the root of the game tree. Hence, following this method, we always expand the tip of the tree with the least penalty, while resolving ties arbitrarily.

Using this technique, the paper proposes a min-max approximation approach, where the penalties calculated as the derivatives of the approximate min-max functions. For example, at a particular node, the value of the node is computed by maximizing/ minimizing the evaluated values of all the successors of that node. However, max/min functions don't provide a meaningful derivative that could be helpful in indicating which node to explore further, other than the min or the max node. To resolve this, the paper suggests using Generalized Mean Values, defined as $M_p(a) = [(1/n)(\Sigma a_n^p)]^{1/p}$,for $a=(a_1, a_2 \ldots a_n)$. This function, for p values tending to infinity and -infinity, provide a

good approximation for max and min values for a set of numbers. More importantly, they can provide more indicative values when differentiated, hence allowing for the ability of being able to choose the tip to be expanded that has least penalty and maximum influence on the game play, and providing a better estimate for the back-propagated value of the ancestral nodes.

However, computing the generalized mean values are computationally expensive. To solve this, the paper proposes an alternative solution of \using the appropriate min-max values for value propagation and only using the derivatives of this function to assign penalties, and choosing the optimal expandable tip of the tree.

The results of this approach were promising and indicated being better than alpha-beta pruning when CPU time was a lesser consideration as compared to expense of calling the “move” function. However, since this was an implementation of a penalty-based system, which needs to store the entire explored game tree in memory, it was not desirable in systems with memory constraints.