Game Tree Searching By Min/Max Approximation by Ronal Rivest

Game search trees can become unwieldy with increasing depth, thus requiring evaluation of an exponentially increasing number of nodes to be evaluated in order to determine the immediate next move. The contribution of this paper is to avoid this exponential explosion by only growing the part of the search tree that is most likely to result in a winning move, rather than expanding the tree at every leaf every time the depth of the search increases. Given a search tree, an alogrithm is described to determine the leaf node that should be searched next, rather than expanding all leaf nodes of the tree. Thus the tree grows at non-exponential rate, more specifically, a tree with each node having an average degree of d, grows by d nodes with each iteration of the algorithm (the max depth of the tree may or may not increase by one with the consideration of each addtional node) -- recall that regular min max requires expanding all leaf nodes when increasing depth by 1.

One of the key ideas in this paper is to use the notion of Generalized Mean Value (GMV) function , M(A,p), where A is a vector of values and p is an integer. GMV has the property that for large positive values of p, M(A,p) approaches the maximum value in the vector A, and for large negative values of p, M(A,p) approaches the minimum value of the vector function. GMV is a continuous (partially) differentiable function with respect to values in A, when p is allowed to take any real value, rather than just integral values. This allows the calculation of derivates of the mean values function at every node C of the tree with respect to all the leaf nodes of the subtree rooted at C. Then, using the chain rule of differentiation, which requires that the function be continuous and differentiable, it is possible to find the differential of the root node of the tree with respect to every child node of the tree. Intuitively, the differentiability of the GMV function allows the determination of how much affect the small change in the value of a child tree will have on the root node of the tree, and then choosing that node as the next node to be expanded. This is the second key observation that is central to the algortihm presented in the paper.

The algorithm describes the notion of a real-valued penalty associated with each leaf node that can be computed based on some heuristic to determine the "score" of that node. Note that the score calculation can be anything including "difference between player moves and opponent moves" as described in the class, as that is not part of the key results in this paper.

As the game progresses, the search tree must be stored in its entirety so that weights and penalties for every node can be recomputed with each move on the board. The weight of each edge in the tree is a negative log function (page 89) of the values of the nodes forming the edge. Based on this weight, the penalty associated with every child of the leaf node that was expanded is computed, and the new penalties of all the nodes propagated to ancestors of these nodes all the way back to the root, thus affecting the penalties associated with all the leaf nodes in the tree. Then the minimum penalty over all leaf nodes is computed in order to choose the leaf node that will be expanded next. Nodes that cannot be expanded are given infinite penalties in order remove them from consideration while expanding the tree.

Summarily, the paper provides a heuristic to expand the search tree in a targeted manner, picking only one leaf node for consideration at each step -- the leaf node that least affects the outcome of game relative to the root. Combined with a good heuristic to determine the score for each node, the search tree grows only in the subtrees that have the highest chances of containing the winning move.