Game Tree Searching by Min / Max Approximation, by Ronald Rivest, describes an approach to searching game trees that is very similar to the Minimax algorithm with iterative deepening, but substitutes a generalized mean-value operator for the max and min operators. The rationale is that the generalized mean operator is continuously differentiable with respect to its arguments, which in this case are the just-expanded successor nodes to a tip of the partial game tree under consideration. Newly expanded nodes are first assigned a value using a static evaluator, but Rivest then uses the partial derivatives to compute penalty terms that are used to select the path to the best outcome.

The generalized mean value operator is defined as:

$$M_p(a) = (\frac{1}{n}\sum_{i=1}^{n} a_i^p)^{\frac{1}{p}}$$

As *p* increases, the result approximates *max(a)* by approach the value of the largest a_i. Similarly, for large negative values of *p*, the result approaches *min(a)*. But as opposed to *max* and *min*, a partial derivative can be calculated for each of the a_i:

$$\frac{\partial M_p(a)}{\partial a_i} = \frac{1}{n}(\frac{a_i}{M_p(a)})^{p-1}$$

The possible value of the derivative ranges between 0 and 1, and Rivest sets the penalty for choosing the path to any succeeding node *c* as the negative natural logarithm of the derivative evaluated for the approximate or estimated (at a tip) value *a*, or a constant value in the case of a terminal node. Using the chain rule for derivatives, the total penalty for any path is the sum of the penalties for each node in the path, starting at the search root, the smaller the better. For each node that is not a tip node, the algorithm stores the node's approximate current value, the "best-path" successor node and the penalty associated with the best successor plus the calculated penalty to get to it (from the derivative).

On each iteration, the algorithm expands the best successor (the root to begin with), applies the appropriate mean-value operator to the successors, calculates the derivative-based penalties and recomputes the penalties back up to the root. This means that each node stores the cumulative penalty from that node to the tip of the partial game tree reached by following the best successor nodes. It can happen when traveling down the tree from a node with a sibling that has a penalty term not much higher, that the active path accumulates a greater penalty than the sibling, at which point the sibling will become the next node to expand. So the algorithm could switch branches any number of times. It is here that an important difference from the minimax algorithm shows. A minimax max node would randomly choose between a move with possible values (30, 40) and (35, 40), because the max operator would return only the largest, but the generalized mean algorithm would favor (35, 40). So the generalized mean approach chooses moves with better secondary moves should the primary move not work out further down the tree.

Results

The author's team set up a Connect Four tournament that pitted their algorithm against minimax with alpha-beta pruning, using the same static evaluation function and controlling for all combinations of the opening two moves. They found that min/max approximation outperformed alpha-beta when the limiting resource was number of function calls (in this case to the move, or forecast, function), but that alpha-beta was superior when the limiting resource was time. In analyzing the performance, they found that alpha-beta called the move operator 3500 times per second, while min/max approximation called it only 800 times per second. Hence, they concluded that min/max approximation would do better than alpha-beta if the function calls needed to implement the game simulation were computationally expensive. They noted, however, that the algorithm was very sensitive to the exact form of the penalty term, and that further research was needed on how to make this computation more robust.