Summary of: Game Tree Searching by Min/Max Approximation*

This paper introduced a new method when evaluating the Min/Max algorithm by using the generalized mean values for estimates of the min and max parts of the calculation. The generalized mean value function is an interesting way to generate an approximation of the minimum and maximum using an input vector. The function is defined as: $M_p(\boldsymbol{a}) = \left(\frac{1}{n}\sum_{i=1}^{n} a_i^p\right)^{1/p}$, where p is any non-zero, real number, and the case for $p = 0$ looks a little different: $M_0(\boldsymbol{a}) = (\prod_{i=1}^{n} a_i)^{1/n}$. For this function the parameter p needs to be defined for computation. For large values of p, the result is a nice approximation of the maximum, and large negative values results in an approximation of the minimum. The interest, though, is in taking the partial derivatives of the generalized mean value functions for each element, $\frac{\partial M_p(\boldsymbol{a})}{\partial a_i} = \frac{1}{n}\left(\frac{a_i}{M_p(\boldsymbol{a})}\right)^{p-1}$, this reveals which leaf the root depends on the most determining which to expand.

The tests performed use a penalty-based search method, where weights are assigned to every edge in the tree, high values for bad moves and low values for good moves. The penalty is sum of all the weights of all the edges between a node and the root. Based on this, the node with the smallest penalty should be picked for expansion. For this paper the penalties are created by using the derivatives from the approximating functions using a large value for p. These approximations represent the static evaluation function estimates for the terminal positions.

For an implementation and comparison, the Mini/Max approximation is compared to the usual Mini/Max algorithm with alpha/beta pruning using games of connect-four. The Mini/Max approximation uses a custom heuristic with the “reverse approximations”, which would be described with the formula for the weights: $w(c) = \log(n) + (p-1)\ \cdot \left(\log\left(\hat{v}_E(d)\right) - \log\left(\hat{v}_E(c)\right)\right)$. This formula comes from using the function $D(x, y) = \frac{\partial \tilde{v}_E(x)}{\partial\ \tilde{v}_E(y)}$ to represent the sensitivity of the root to changes in the tip, and $w(x) = -\log\left(D(f(x), x)\right)$ to be the weight on the edge between $f(x)$ and x. Using the derivative from the generalized mean values functions $w(c) = -\log\left(D(d, c)\right) = -\log\left(\frac{1}{n}\left(\frac{\hat{v}_E(d)}{\hat{v}_E(c)}\right)^{p-1}\right) = -\log(1) + \log(n) + (p-1)\ \cdot\left(\log\left(\hat{v}_E(d)\right) - \log\left(\hat{v}_E(c)\right)\right) = \log(n) + (p-1)\ \cdot\left(\log\left(\hat{v}_E(d)\right) - \log\left(\hat{v}_E(c)\right)\right)$, c is any node in some subtree rooted at d. In the implementation $\log(n)$ is replaced with the value 0.05 based on preliminary testing, $\hat{v}_E(d)$ is the value of the node, and $\hat{v}_E(c)$ is the value of the sibling of d with the best backed up score. There were 98 total games played, 49 with the alpha/beta agent moving first and 40 where the min/max approximation going first. Each group of 49 games was created by dropping the first two pieces on the 7-column grid ($7 \times 7 = 49$ combinations). For one set of trials iterative deepening was performed at different time bounds ranging from one to five seconds. The second trial was based on the number of moves made, ranging from 1,000 to 5,000. The results show that for iterative deepening based on CPU time, that the alpha/beta pruning method performed better for all times tested, and for iterative deepening based on moves made the Mini/Max method outperformed alpha/beta pruning for all levels tested.