Part 2: Research Review

Selected Paper: Game Tree Searching by Min / Max Approximation

Paper's goals or techniques or ideas introduced:

Idea 1

The conventional min or max functions have discrete results, the result can be only one of the input values. Instead if we use an averaging function **M**, which we can adjust the sensitivity with an additional parameter, we can get continuous result space (by adjusting the sensitivity factor **p**).

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

This type of function has continuous derivatives and derivatives tells us about the rate of change of the original function. If we take the partial derivative of the original function for each input value **a,** we can determine which child state has the most dramatic change in score. A positive high **p** value behaves like a maximizing function where a negative high **p** behaves like a minimizing function.

Idea 2

Iterative search means we pick the next node to expand by evaluating some combination of properties in the explored search tree.

To select the node to expand, we can assign ‘weights’ (penalties) to the each edge, such that edges leading to a bad move is penalized more. A possible formula for assigning the weight can include factors like:

- the penalty for descending one level in the search tree
- how good or bad the candidate node is compared to it’s siblings

Then we can select the node with the smallest penalty to expand next.

Techniques

The “min/max approximation” heuristic explained in the paper is rooted mainly on these two ideas. It is a penalty-based search method and the penalties are calculated according to the derivatives of approximating functions. Next node with the minimum penalty is selected to expand next.

The drawback of this algorithm is the computational difficulty of computing the mean for large **p** values. Large **p** values increases accuracy but also increases the runtime of the calculation.

Paper's results & learnings:

The algorithm is tested on a game of Connect4:

For two different resource bounds (CPU time in seconds & number of calls to the “move” subroutine in thousands) the results are:

TABLE 2. Experimental results

Resource bound per turn	MM wins	AB wins	Ties
1 second	41	46	11
2 second	40	42	16
3 seconds	36	44	18
4 seconds	39	52	7
5 seconds	30	55	13
Total	186	239	65
1000 moves	47	35	16
2000 moves	50	35	13
3000 moves	42	47	9
4000 moves	49	42	7
5000 moves	61	31	6
Total	249	190	51

- Based on time usage, minimax search with **A**lpha-**B**eta pruning was superior and based on number of calls made to the “move” subroutine, **M**in/**M**ax Approximation was superior.
- The AB implementation was calling the “move” operator approximately 3500 times per second while MM was only calling it 800 times per second.

- Penalty based schemes may not perform well unless they are given a large amount of memory to work with, because the tree being explored is explicitly stored.
- Penalty based schemes presented in the paper require that a tip be expanded by generating and evaluating all of the tip's successors. Many search schemes are able to skip the evaluation of some of the successors in many cases.
- Penalty based schemes may appear inefficient (compared to depth first search schemes) since they spend a lot of time traversing back and forth between the root and the leaves (updating the cumulative weights and selecting the min weighted node from the root level)