Introduction

This is a short review of "Game Tree Searching by Min/Max approximation" by Ron Rivest

Key question

The main question the paper resolves around is: Given a partial game tree E, how do you select which node you should "extend"

In the classical alpha beta pruning search with iterative deepening we do follow an "expand all" strategy from depth d to depth d+1. The question is can we do better and selectively expand node and if yes, is it efficient?

This is the main subject of the paper. There is also one more issue that arises in the discussion of the paper that will be addressed at the end.

Formulation

A big part of the paper is focused in the precise formulation of the tree, the partial tree, the definition of the evaluation function etc...

- The basic idea is we want to find which leaf nodes to extend
- for that reason we assign to every edge (from parent to child) a weighting factor
- At every leaf node we can sum the penalties leading from the root to that node.
- The leaf node with the least penalty is the one to be extended

Selecting the weighting function

- We want to expand that tip node that the root is most "sensitive" to its change
- if we used as w(c) = |eval(c) - best(parent(c))| then we we always expand along the principal variation because it will always have a zero penalty.
- to avoid that, we use a generalized notion of the min and max functions, borrowed from L^p spaces
- we define a generalized mean for every p as follows :

$$M_a^p = \left(\frac{1}{n} * \sum a_i^p\right)^{1/p}$$

- if we let p go to +inf then what we get is the maximum function and if we let p go to -inf then what we get is the minimum function. The benefit of this formulation is that we can now have continuous approximations of the min an max functions and this calculate derivatives
- the partial derivate of the new function M on one of its components is:

$$\frac{\partial M_a^p}{\partial a_i} = \frac{1}{n}\left(\frac{a_i}{M_a^p}\right)^{p-1}$$

- if we take the derivative of the max function computed on a 'father' node (denoted as f(c)) against one of its children we have (where d is the sibling of c with the best evaluation)

$$\frac{\partial M^p(f(c))}{\partial c} = \frac{1}{n}\left(\frac{eval(c)}{M^p(f(c)))}\right)^{p-1} = \frac{1}{n}\left(\frac{eval(c)}{eval(d)}\right)^{p-1}$$

- if we take logs we have
  - $-[log(1/n) + (p-1) * log(eval(c)) - log(eval(d))]$
    - this last expression quantizes how much a change in node c will affect the parent f(c)
    - the **negative** of this exact expression we set at the weight w(c) and attach it to the edge from f(c) to c. N in this equation is the number of siblings of c
- because we took the negative of the derivative and we want the biggest influence , we are searching for the tip node with the least penalty (defined above as the sum of weight of the path leading to c)
- choosing a large value of p, thus approximating the max function will result in selecting the "best" moves to expand , small values of p, will result in a wider tree and more nodes expanding.

Remarks

- The strategy was tested in the game "Connect Four".

The results are that

- it takes fewer steps to get to a better results
- it takes more time to get to that better result.