An implementation of the Min/max approximation approach

Goal:

The writer's goal is to use a new technique for searching in game trees, based on the idea of approximating the min and max operators with generalized mean-value operators. The approximation is used to guide the selection of the next leaf node to expand, which is expected to have the largest effect on the value

By using the generalized mean values to approximate the min and max functions, the writer can identify in leafs in a game tree upon whose value the value at the root depends most strongly. This is done by taking derivatives of the generalized mean value functions at each node and using the chain rule. This leaf will be the one to expand next.

Generalized p-mean of a, $M_p(a)$ can be calculated by:

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

Let $a = (a_1, \ldots., a_n)$ be a vecter of n positive real number, and let p be a non-zero real number.

In case $p = 0$:

$$M_0(a) = \lim_{p\to 0} M_p(a) = (a_1 a_2 a_3 \ldots a_n)^{1/n}$$

The partial derivative of $M_p(a)$ with respect to a_i is:

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

The proposed technique requires a single static evaluator $\hat{v}(\cdot)$ whilst some other methods require two static evaluation functions which are upper and lower bonds on $v(\cdot)$

For any partial game tree E and any $c \in E$, to calculate the backed-up estimates $\hat{v}_E(c)$ of $v(c)$. I think the equation is the same as Minimax algorithm:

$$\hat{v}_E(c) = \begin{cases} \hat{v}(c), & \text{if } c \in T(E), \\ \max_{d\in S(c)} (\hat{v}_E(d)), & \text{if } c \in \text{Max}/T(E), \\ \min_{d\in S(c)} (\hat{v}_E(d)), & \text{if } c \in \text{Min}/T(E), \end{cases}$$

$T(E)$is a set of terminal game state of E, $S(c)$ are a set of successor nodes of c

The writer assumes that for each tip of $c \in E$ an estimate $\tilde{v}_E(c)$ of $v(c)$ is available wherein "max" has been approximated by "M_p", and 'min' has been approximated by "M_{-p}"

$$\tilde{v}_E(c) = \begin{cases} \hat{v}(c), & \text{if } c \in T(E), \\ M_p\left(\tilde{v}_E(d_1), \ldots, \tilde{v}_E(d_k)\right), & \text{if } c \in \text{Max}/T(E), \\ M_{-p}\left(\tilde{v}_E(d_1), \ldots, \tilde{v}_E(d_k)\right), & \text{if } c \in \text{Min}/T(E), \end{cases}$$

The heuristic proposed in this paper is an iterative technique (Not the classic AB pruning + iterative deepening). The writer lets $w(c)$ denote the weight on the edge between c and its father $f(c)$ and define $w(s)$ to be zero; where s is a root node of a patial tree E. The paper assumes that $w(c)$ is computable from $\tilde{v}_E(f(c))$, and the values $\{\tilde{v}_E(d) | d \text{ is a sibling of } c\}$. (c is also its own sibling.)

The writer defines the "penalty" $P(c)$of a tip $c \in T(E)$ to be the sum of the penalties of all the edges between c and the root s:

$$P(c) = \sum_{d \in A(c)} w(d)$$

Where $A(c)$ is a set of ancestors of c.

For any node $c \in E$, the paper defines $b(c)$ to be the expandable tip node in the sub tree E_c which minimise $P_c(x)$, if none of E_c's tip node are expandable, $b(c) = \omega$

For any node $c \in E$, $a(c)$can be defined as follows. If $b(c) = \omega$ then $a(c) = \omega$, if c is an expandable tip of E then $a(c) = c$

With each node c of E they store $\tilde{v}_E(c), a(c)$ and $\pi(c) = P_c(b(c))$,the penalty of the best expandable tip $b(c)$ of E_c relative to the sub tree E_c, or else ∞ if $b(c) = \omega$; and $P_c(b(c) = 0$ if $b(c) = c$

A penalty-based algorithm begins with $E = \{s\}, a(s) \text{and } \pi(s) = 0$. At each step of the iterative expansion procedure, the following steps are performed:

Step 1. If $b(s) = \omega$ stop -- the tree has no expandable tips (i.e. $E = C$).

Step 2. Set x to s.

Step 3. Whilst $a(x) \neq x$, set $x \leftarrow a(x)$. (Now x is the expandable tip node which minimizes $P(x)$.)

Step 4. Add $S(x)$ to E.

Step 5. Compute $\tilde{v}_E(d)$ for all $d \in S(x)$. For each expandable child d ofx, initialize $a(d)$ to d and $\pi(d)$ to 0. For each terminal child d ofx, initialize $a(d)$ to ω and $\pi(d)$ to be ∞.

Step 6. Recompute $\tilde{v}_E(x), a(x)$ and $\pi(x)$ from the corresponding values at x's children.

Step 7. If $x = s$ stop, otherwise set $x = f(x)$ and go back to Step 6.

When the algorithm terminates in the last step, then it has traced a path from the root s down to the best expandable tip $x = b(s)$ in E_s, added all the successors of x to E, and updated the$\tilde{v}_E$, a, and π values where necessary by a traversal back up the tree from x to the root s.

With the algorithm earlier, the agent can explore a partial game tree by heuristics with penalty based weigh by finding the largest $D(s,x)$ from the least Penalty $P_s(x)$:

$$D(x,y) = \frac{\partial \tilde{v}_E(x)}{\partial \tilde{v}_E(y)}$$

By the chain rule for derivatives, we have:

$$D(s,x) = \prod_{c \in A(x)} D(f(c),c)$$

So that tip x with the largest value $D(s,x)$ is the one with the least total penalty $P_s(x)$

$$P_s(x) = \sum_{c \in A(x)} w(c)$$

Fig. 2 shows a partial tree of size 5; Let's assume all tip of E are expandable. Squares are MAX nodes c and Circles are MIN nodes c, and value $\hat{v}_c$ given to each node c.

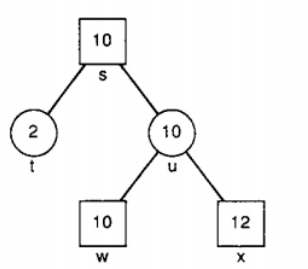

FIG. 2. A partial game tree.

Redrawing Fig. 2 as Fig. 4, using the approximations with $p = 10$. Each node is labelled inside with $\tilde{v}_E(c)$. Each edge from a configuration c to one of its children d is labelled with $w(d)$ on the edges between c and the root s.

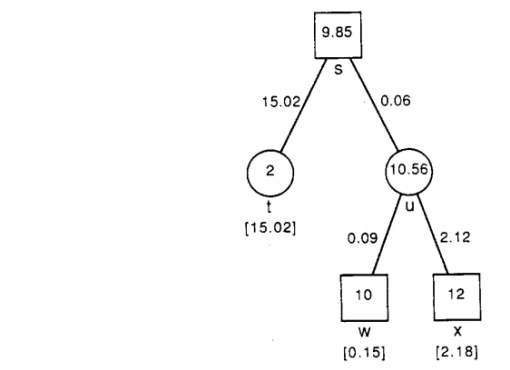

FIG. 4. Game tree with min/max approximations.

According to the rule, w is the best node to expand, since the value at the root depends most strongly on the value at w. (The least penalty $P_s(w)$.)

Redrawing Fig. 4 as new Fig. 5, where $\tilde{v}_E(c)$ is drawn inside node c as before, and the pair $[a(c), \pi(c)]$ is placed to the right of node c. Similarly, Fig. 6 is our revised version of Fig. 5, after w has been expanded. In both figures the edges are labelled with the $w(c)$ values as in Fig. 4

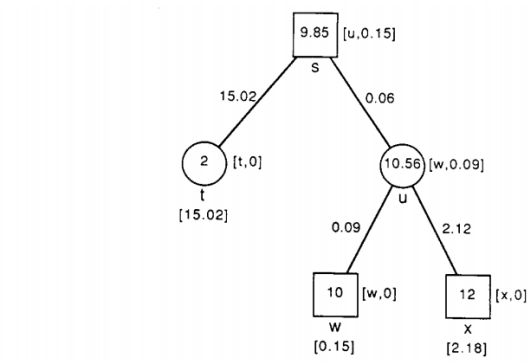

FIG. 5. Game tree with a and π values.

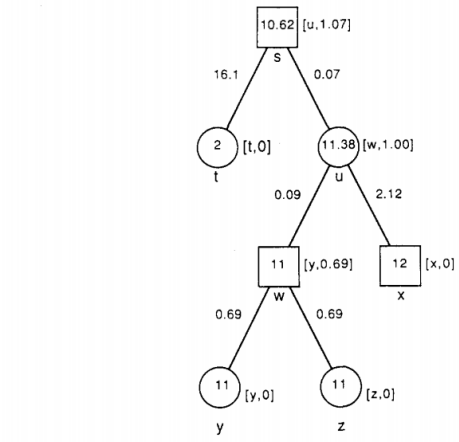

FIG. 6. Expanded game tree with a and π values.

The writer hasn't discovered how to "optimally" choose the parameter p yet. He says that choosing a large value of p corresponds to having a high degree of confidence in the accuracy of the values returned by the static evaluator, while a small p corresponds to a low degree of confidence. Because for small p, the heuristic should grow rather broad trees and vice-versa for a large p.

Result:

The approach in the paper can produce play superior to that produced by minimax search with Alpha-beta pruning, for the same number of calls to the underlying "move" operator. However, when CPU time rather than calls to the move operator is the limiting resource, Minimax search with Alpha-beta pruning seems to play better

Based on time usage alone, Alpha-beta seems to be superior to the implementation of the Min/max approximation approach. The paper says, if the comparison based on move-based resource limits, the story is reversed: Min/max approximation is definitely superior.

Because the Alpha-beta pruning called the move operator approximately 3,500 times per second, whilst the implementation of the min/max heuristic called the move operator approximately

800 times per second. Also the penalty-based schemes are oriented towards improving the value of the estimate $\tilde{v}_E(s)$ at the root, rather than towards selecting the best move to make from the root.

The penalty-based schemes as presented 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.

Finally, the writer observes that penalty-based schemes do spend some time evaluating non-optimal lines of play. However, the time spent examining such lines of play decreases as the number of non-optimal moves in the line increases, according to the weights assigned to those non-optimal moves.