LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-311

GAME TREE SEARCHING BY MIN/MAX APPROXIMATION

RONALD L. RIVEST

SEPTEMBER 1986

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

1 Introduction

This paper introduces a new technique for searching in game trees, based on the idea of approximating the min and max operators with generalized mean-value operators.

Game playing by computer has a long history, and many brilliant ideas have led us to the point where high-quality play for many games can be obtained with pocket-sized computers. (See [BF81] for an exposition of previous work in this area.)

However, further improvement is certainly possible, and this area of research is still an active one. The combinatorial explosion of possibilities in a game such as chess tax our most powerful computers, and even special-purpose hardware soon reaches its limits. Clearly, the most careful organization and allocation of computational resources is needed to obtain expert-level play. Techniques such as alpha-beta pruning and its successors [KM75, Be79] have been essential in reducing the computational burden of exploring a game tree. Still, new techniques are needed. Nau et. al. [NPT85], after much expermentation with existing methods, assert that "A method is needed which will always expand the node that is expected to have the largest effect on the value". This paper suggests such a method.

The method proposed in this paper, "min/max approximation", attempts to focus the computer's attention on the important lines of play. The key idea is to approximate the "min" and "max" operators with generalized mean-value operators. These are good approximations to the min/max operators, but have continuous derivatives with respect to all arguments. This allows us to define the "expandable tip upon whose value the backed-up value at the root most heavily depends" in a non-trivial manner. This tip is the next one to be expanded, using our heuristic.

We also describe some preliminary experimental validation of these ideas.

In section 2 of this paper I present the essential results about generalized mean values that underly the new method. Then in section 3 these ideas are applied to the problem of searching game trees. In section 4 I give some thoughts regarding implementation details. Section 5 describes our preliminary experimental results. Some final thoughts are presented in section 6.

2 Generalized Mean Values

Let $\mathbf{a} = (a_1, \ldots, a_n)$ be a vector of n positive real numbers, and let p be a non-zero real number. Then we can define the generalized *p-mean* of $\mathbf{a}$, $M_p(\mathbf{a})$, by

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

p	-32	-16	-8	-4	-2	-1	0
$M_p(\mathbf{a})$	10.4	10.9	11.9	13.9	16.6	18.7	21.0

p	1	2	4	8	16	32
$M_p(\mathbf{a})$	23.0	24.5	26.5	28.3	29.7	30.7

Table 1: $M_p(\mathbf{a})$ for $\mathbf{a} = (10, 21, 29, 32)$

Of course, $M_1(\mathbf{a})$ is the ordinary arithmetic mean. We can extend our notation to the case $p = 0$ by

$$M_0(\mathbf{a}) = \lim_{p \to 0} M_p(\mathbf{a}) = (a_1 \cdots a_n)^{1/n} \tag{2}$$

so that $M_0(\mathbf{a})$ is the geometric mean.

We begin with the fact that

$$p < q \Rightarrow M_p(\mathbf{a}) \leq M_q(\mathbf{a}) \tag{3}$$

where equality only holds on the right if all the a_i's are equal. (See [HLP34] for proofs and other facts about generalized mean values.)

For our purposes, we are most interested in the following two facts:

$$\lim_{p \to \infty} M_p(\mathbf{a}) = \max(a_1, \ldots, a_n) \tag{4}$$

$$\lim_{p \to -\infty} M_p(\mathbf{a}) = \min(a_1, \ldots, a_n) \tag{5}$$

To illustrate the above facts, consider Table 1 where various values of $M_p(\mathbf{a})$ are given for $\mathbf{a} = (10, 21, 29, 32)$ and various p.

For large positive or negative values of p, $M_p(\mathbf{a})$ is a good approximation to $\max_i(a_i)$ or $\min_i(a_i)$, respectively. However, $M_p(\mathbf{a})$, unlike max or min, has *continuous* derivatives with respect to each variable a_i. The partial derivative of $M_p(\mathbf{a})$ with respect to a_i is

$$\frac{\partial M_p(\mathbf{a})}{\partial a_i} = \frac{1}{n} \left(\frac{a_i}{M_p(\mathbf{a})} \right)^{p-1} \tag{6}$$

The major reason that the generalized means are of interest to us here is that they are more suitable for a "sensitivity analysis" than the min or max functions. We propose that $\partial M_p(\mathbf{a})/\partial a_i$ (for large p) is a more useful quantity than $\partial \max(\mathbf{a})/\partial a_i$, since the latter is zero unless a_i is the maximum, in which case it is one. This discontinuous behavior is

awkward to work with, whereas the derivative (6) is continuous. We also note that the derivative (6) ranges in value from 0 to $n^{-1/p} \approx 1$ for $p \gg \ln n$).

By way of example, with $\mathbf{a} = (a_1, a_2, a_3, a_4) = (10, 21, 29, 32)$, and $\mathrm{p} = 32$, we have $\nabla M_p(\mathbf{a}) \cong (2 \times 10^{-16}, 2 \times 10^{-6}, 0.04, 0.90)$. The a_i values near the maximum have much more effect on $M_p(\mathbf{a})$ here than do smaller values.

We observe as well that other forms of generalized mean values exist. If f is any continuous monotone increasing function (such as the exponential function), we can consider mean values of the form

$$f^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} f(a_i)\right). \tag{7}$$

Using $f = \exp(\cdot)$ yields a good approximation to $\max(\ldots)$, and $f = \ln(\cdot)$ yields a good approximation to $\min(\ldots)$. We shall not use these approximations in this paper.

3 Game Tree Searching

3.1 Game Trees

We consider a two-person zero-sum perfect information game between two players, Min and Max. The game specifies a finite tree C of *configurations* with a distinguished *start configuration* $s \in C$. The game begins in configuration s with Max to move; after that they alternate turns. We partition the set C into the disjoint subsets *Min* and *Max* of those configuration for which it is Min's turn to move, and those configurations for which it is Max's turn to move. The game defines for each $c \in C$ the set $S(c)$ of c's *successors* (or *children*). The player to move from configuration c selects some $d \in S(c)$ as his move; his opponent must then move from configuration d on his turn. Those configurations which have no successors are called *terminal configurations*; we denote by $T(C)$ the set of terminal configurations. The game stops when a terminal configuration is reached. We assume that the relation S is acyclic, so that the game is guaranteed to terminate. The game thus has the structure of a tree with vertex set C, root s, and leaf set $T(C)$. The actual play traces out a path in the tree from the root s to leaf representing a terminal configuration t.

Each leaf $t \in T(C)$ has an associated *value* or *score* $v(t)$; this is the value of the terminal position from Max's point of view (i.e. the amount Min must pay Max if the game terminates in configuration t). By induction on the structure of the tree we may determine the value $v(c)$ of any configuration $c \in C$ by "backing up" or "minimaxing" the values at the leaves:

$$v(c) = \begin{cases} v(c) & \text{if } c \in T(C) \\ \max\limits_{d \in S(c)} v(d) & \text{if } c \in Max \setminus T(C) \\ \min\limits_{d \in S(c)} v(d) & \text{if } c \in Min \setminus T(C) \end{cases} \tag{8}$$

Given $v(c)$ for all $c \in C$, optimal moves are easy to determine: if $c \in Max$, then player Max should select any configuration $d \in S(c)$ with maximum value $v(d)$. Similarly, if $c \in Min$, then player Min should select any configuration $d \in S(c)$ with minimum value $v(d)$. Figure 1 shows a small game tree. Configurations in *Max* are shown as squares, those in *Min* are shown as circles. The value of each configuration is shown inside the square or circle. The value of the game is 16; optimal play is indicated with the double arrows.

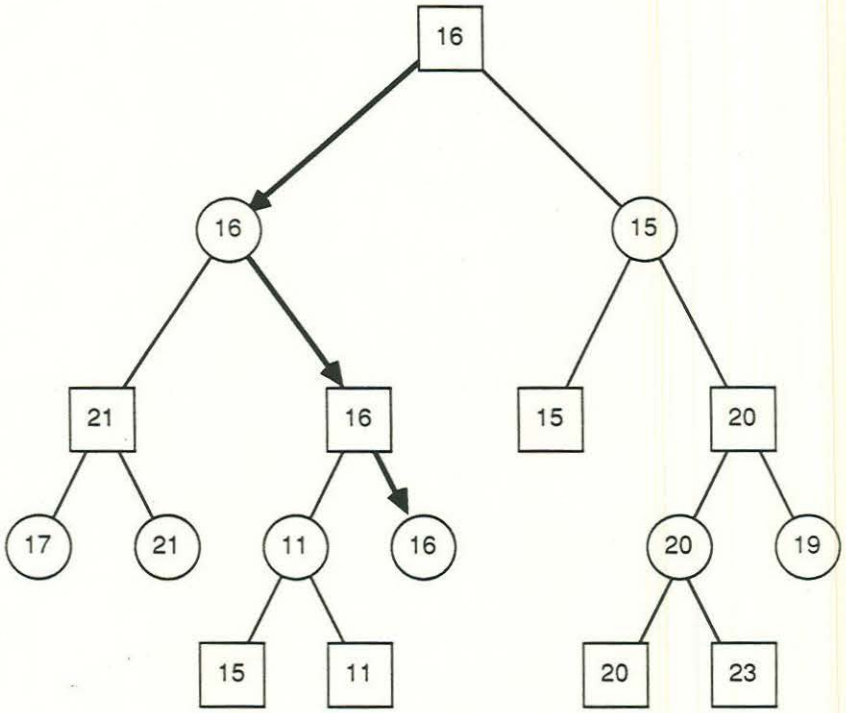

Figure 1: A small game tree

3.2 Searching a Game Tree

When C is sufficiently small, the game tree can be explored completely, and optimal play is possible. In some cases a "pruning" heuristic such as alpha-beta [KM75] is sufficiently effective that optimal play is possible, even though only a small fraction of the game tree is explored – the portions of the tree that are "pruned" (not explored) are known not to be relevant, so that at least implicitly the entire tree is explored.

However, for most interesting games the game tree is so large that complete exploration (even implicitly) is impossible, and heuristic approximations need to be employed.

A heuristic method is usually based on a "static evaluation function" $\hat{v}$ that provides an estimate $\hat{v}(c)$ of the backed-up value $v(c)$ for *non-terminal* nodes c. Such a static evaluation is based on various "static" features of the board that can be evaluated without further look-ahead (e.g. piece count and the advantage for the player to move).

The technique proposed in this paper only requires a single static evaluator $\hat{v}(\cdot)$. By comparison, some other methods – most notably the B* algorithm [Be79] – require *two* evaluation functions which return respectively upper and lower bounds on $v(\cdot)$.

A popular approach to handling very large game trees is to select a suitable depth-cutoff d, estimate the values of nodes at depth d using the static evaluator, and to compute the backed-up value from the nodes at depth d using the alpha-beta heuristic.

Given a limit on the computing time available, one can apply the above idea by successively computing the backed-up values for depths $d = 1, 2, \ldots$, until one runs out of time. This technique is known as *progressive deepening.*

As d increases one would hope that the accuracy of the value computed for the root would improve – searching deeper should yield increased accuracy. This seems to be true for common games such as chess, but there are other "pathological" games for which searching deeper seems to yield *less* accuracy [Na82, Na83]. We assume for this paper that our game is non-pathological.

A different class of heuristics are known as *iterative* heuristics. These heuristics "grow" the search tree one step at a time. At each step one tip node (or leaf) of the current search tree is chosen, and the tree is expanded by adding the successors of that tip node to the search tree. Examples of such iterative techniques are the Berliner's B* algorithm [Be79], Nilsson's "arrow" method [Ni68], Palay's probability-based method [Pa85], and McAllester's "conspiracy number" method [Mc86]. After each step, the values provided by the static evaluator at the newly added leaves are used to provide new backed-up values to the ancestors of these leaves in the tree. The tree grown by an iterative heuristic may not be of uniform depth: some branches may be searched to a much greater depth than other branches.

The heuristic proposed in this paper is an iterative technique in the above sense.

3.3 Iterative Search Heuristics Details

We now formalize how a tree is explored using an iterative search heuristic, and how estimates are "backed up" to the root.

We begin with some straightforward definitions.

If $d \in S(c)$, we say that c *is the father of* d. In general, we denote the father of d by $f(d)$, and define $f(s) = s$. We call c a *sibling* of d if $f(c) = f(d)$. Also, we call c an *ancestor*

of d if $c = d$ or if c is an ancestor of $f(d)$. We let $A(d)$ denote the set of d's ancestors. Note that c is both an ancestor and a sibling of itself, for any c.

We say that a subset $E \subseteq C$ *is a partial game tree* if, for any $d \in E$, all of d's siblings and ancestors are in E.

If E is a partial game tree, then for any $c \in E$ the *subtree of E rooted at c*, denoted E_c, is defined to be $\{x \mid x \in E \& c \in A(x)\}$, the set of all $x \in E$ which have c for an ancestor.

If E is a partial game tree, then $c \in E$ is a *tip of E* if c has no successors in E, i.e. $S(c) \cap E = \emptyset$. The set of tips of E is denoted $T(E)$. The tips of C are the terminal configurations. We say that a tip x of E is *expandable* if x has successors.

If E is a partial game tree, and c is an expandable tip of E, then to *expand E at c* means to add the successors $S(c)$ of c to E. If we expand E at a tip c, then E remains a partial game tree.

We let $\hat{v}$ denote our static evaluation function. We assume that $\hat{v}(c) = v(c)$ if $c \in T(C)$, i.e. that our static evaluation function is exact on the set of terminal positions. For other positions c the static evaluation $\hat{v}(c)$ is merely an estimate of the correct backed-up value $v(c)$. We make no particular assumptions about the relationship between $\hat{v}(c)$ and $v(c)$.

For any partial game tree E and any $c \in E$ we can define the backed-up estimates $\hat{v}_E(c)$ of $v(c)$ in a manner identical to the way the correct values are backed-up using equation (8), except that we are backing up the static evaluations from the tips of E rather than backing them up from the terminal positions of C:

$$\hat{v}_E(c) = \begin{cases} \hat{v}(c) & \text{if } c \in T(E) \\ \max\limits_{d \in S(c)} (\hat{v}_E(d)) & \text{if } c \in Max \setminus T(E) \\ \min\limits_{d \in S(c)} (\hat{v}_E(d)) & \text{if } c \in Min \setminus T(E) \end{cases} \tag{9}$$

If $E = \{s\}$ (our partial game tree is the minimal possible game tree), then $\hat{v}_E(s)$ is just $\hat{v}(s)$, the static evaluation at s. On the other hand, if $E = C$ (our partial game tree is the complete game tree), then our estimates are exact: $\hat{v}_E(c) = v(c)$ for all c. We expect that $\hat{v}_E(s) \rightarrow v(s)$ as $E \rightarrow C$ for our "non-pathological" game. (The use of the limiting notation "$\rightarrow$" here is merely suggestive and not formal.)

We observe that if we expand E at c, then we can update $\hat{v}_E$ by recomputing equation (9) only at nodes in $A(c)$ (first at c, then $f(c)$, and on up the tree until s is reached).

The general process of partially exploring a game tree by an iterative heuristic can be formalized as follows:

1. Initialize E to $\{s\}$, and $\hat{v}_E(s)$ to $\hat{v}(s)$.

2. While $E \neq C$, and while time permits, do:

2a. Pick an expandable tip c of E.

2b. Expand E at c.

2c. Update $\hat{v}_E(c)$ at c and the ancestors of c up to the root s, using equation (9).

The major unspecified detail here is in step 2a — which expandable tip c of E should we pick? The purpose of this paper is to provide a new answer to this question.

Another intriguing question is how one might gauge the accuracy of the current estimate of the value at the root, in case one wishes to use a termination condition based on accuracy instead of a termination condition based on time. We do not pursue this question is this paper.

3.4 Penalty-Based Iterative Search Methods

We now present a general framework for choosing which leaf to expand in an iterative method.

We start by assuming that for each node $c \in E$ an estimate $\tilde{v}_E(c)$ of $v(c)$ is available. We may have $\tilde{v}_E(c) = \hat{v}_E(c)$ (the backed-up value for the root of E_c), or $\tilde{v}_E(c)$ may be defined in some other manner. We assume that if c is not a tip of E, then $\tilde{v}_E(c)$ is determined from the estimates $\tilde{v}_E(d)$ where d is a child of c, and the knowledge of whether $c \in Min$ or $c \in Max$. If c is a tip of E, we assume that $\tilde{v}_E(c) = \hat{v}(c)$. The reason for introducing the estimation function $\tilde{v}_E()$, instead of just using the backed-up estimates $\hat{v}_E()$, is that our method requires this extra generality — we do not necessarily use the straightforward backed-up values as our estimates, although our estimates $\tilde{v}_E()$ will also be computable in a "bottom-up" manner.

We assume that it is possible to assign a "weight" to each edge in the current game tree. We expect that edges representing bad moves will be weighted (i.e. penalized) more heavily than edges representing good moves. We denote the weight on the edge between c and its father $f(c)$ by $w(c)$; this may be an arbitrary real number. For convenience we also define the "weight" $w(s)$ (of the nonexistent edge into the root) to be zero.

We assume that the weight $w(c)$ is computable from $\tilde{v}_E(f(c))$, and the values $\{\tilde{v}_E(d) \mid d \text{ is a sibling of } c\}$. (Recall that c is a sibling of itself.) As a concrete example, we might define $w(c)$ to be $\alpha + (\tilde{v}_E(d) - \tilde{v}_E(c))^2$, where $\alpha > 0$ is the penalty for descending a level and where d is the sibling of c which optimizes $\tilde{v}_E(d)$. We shall use a different weight function in our actual proposal, but this should give you an idea of the sort of weight functions that might be suitable.

We then define the "penalty" $P(c)$ of a tip c of E to be the sum of the weights of all the edges on the path connecting c to the root s of the tree: $P(c) = \sum_{d \in A(c)} w(d)$.

The idea is then to expand that tip node t which has the *least* penalty $P(t)$. After t's children are added to the current search tree, we may need to update the estimates $\tilde{v}_E(c)$

of all the ancestors c of t, and we may need to update the weights associated with the edges between an ancestor of t and its children.

The min/max approximation technique presented here is such a penalty-based scheme.

We now present the penalty-based scheme in more detail.

Let E denote the current partial game tree.

For any node $c \in E$, and any node $d \in E_c$, we define $P_c(d)$ to be the sum of the weights of the edges on the path connecting d to c. Here $P_c(d)$ is the penalty of node d relative to the subtree rooted at c. (So the notation $P(d) = P_s(d)$ is just a special case.)

For any node $c \in E$, we define $b(c)$ to be that "best" expandable tip node in the subtree E_c rooted at c, in the sense that $x = b(c)$ minimizes $P_c(x)$ over all expandable tip nodes in E_c. Ties are resolved arbitrarily (e.g. by selecting the "leftmost" such x). If E_c contains no expandable tip nodes, we define $b(c)$ to be the special value ω.

For any node $c \in E$, we define $a(c)$ as follows. If $b(c) = \omega$ then $a(c) = \omega$. If c is a an expandable tip of E then $a(c) = c$. Otherwise $a(c)$ is defined as that child $d \in S(c)$ such that $b(c)$ is an element of E_d. We can think of $a(c)$ in general as an "arrow" from c to one of its children, such that by following successive arrows we are eventually led from c to $b(c)$. (See [Ni68] for the origin of this "arrow" terminology.)

With each node c of E we store the following three values:

1. $\tilde{v}_E(c)$, our estimate of $v(c)$,

2. $a(c)$, the "arrow" from c to its child that contains $b(c)$ (or c if it is an expandable tip or else ω if E_c contains no expandable tips),

3. $h(c) = P_c(b(c))$, the penalty of the best expandable tip $b(c)$ of E_c relative to the subtree E_c, or else ∞ if $b(c) = \omega$. (Note that $P_c(b(c)) = 0$ if $b(c) = c$ is an expandable tip node.)

The weight $w(c)$ need not be stored, since we assumed that it is computable from $\tilde{v}_E$ applied to the father of c and to the siblings of c.

We note that one can compute the values $\tilde{v}_E(c)$, $a(c)$, and $h(c)$ from the corresponding values for the children of c. Specifically, $\tilde{v}_E(c)$ is computable in this manner by assumption, $a(c)$ is the child d of c which maximizes $h(d) + w(d)$, and $h(c)$ is $h(a(c)) + w(a(c))$. (If all the $b(d_i)$'s are ω, then $a(c) = \omega$ and $h(c) = \infty$.)

A penalty-based algorithm begins with $E = \{s\}$, $a(s) = s$, and $h(s) = 0$.

At each step of the iterative expansion procedure, the following steps are performed:

1. If $b(s) = \omega$ stop – the tree has no expandble tips (i.e. $E = C$). Otherwise continue on to the next step.

2. Set x to s, the root of the tree.

3. While $a(x) \neq x$, set $x \leftarrow a(x)$. (Now x is the tip node with minimum penalty $P(x)$.)

4. Add the sucessors of x to E.

5. Compute $\tilde{v}_E(d)$ for all successors d of x. For each expandable successor d of x, initialize $a(d)$ to be d and $h(d)$ to be 0. For each terminal successor d of x, initialize $a(d)$ to be ω and $h(d)$ to be ∞.

6. Recompute $\tilde{v}_E(x)$, $a(x)$, and $h(x)$ from the corresponding values at the successors of x.

7. If $x = s$ stop, otherwise set $x = f(x)$ and return to the previous step.

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 h values where necessary by a traversal back up the tree from x to the root s.

3.5 Discussion of Penalty-Based Schemes

Before getting into the details of our Min/Max approximation method, it is worthwhile discussing some of the general features of penalty-based schemes.

First, we note that penalty-based schemes – like all iterative schemes – require that the the tree being explored be explicitly stored. Unlike depth-first search schemes (e.g. alpha-beta), penalty-based schemes may not perform well unless they are given a large amount of memory to work with.

Second, we note that 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. For example, if there is only one move to make from the root, then a penalty-based scheme may search the subtree below that move extensively, even though such exploration can't affect the decision to be made at the root. By contrast, the B* algorithm [Be79], another iterative search heuristic, is oriented towards making the best choice, and will not waste any time when there is only one move to be made.

Third, 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.

Fourth, we note that penalty-based schemes may appear inefficient compared to depth-first schemes (e.g. alpha-beta), since the penalty-based schemes spend a lot of time traversing back and forth between the root and the leaves of the tree, whereas a depth-first approach will spend most of its time near the leaves. We imagine that the penalty-based schemes could be adapted to show similar efficiencies, at the cost of not always selecting

the globally least-penalty tip to expand. The algorithm would be modified in step 7 to ascend to its successor some of the time and to *redescend* in the tree by returning to step 3 in the other cases. (However, if $b(x) = \omega$ the algorithm must ascend.) The decision to redescend may be made probabilistically, perhaps as a function of the depth of x, or the change noted so far in $h(x)$. For example, one might continue to redescend from the node x found in step 3 until the number of leaves in E_x exceeds the depth of x. We have not explored these alternatives.

Finally, we observe 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.

3.6 Searching by Min/Max Approximation

The "Min/Max Approximation" heuristic is special case of the penalty-based search method, where the penalties are defined in terms of the derivatives of the approximating functions.

Consider the partial game tree of Figure 2. Here we have a tree E of size 5 that we have explored. The tips of E are all assumed to be expandable. The value $\hat{v}_E(c)$ is given at each node c, based on the estimates $\hat{v}(t) = 2$, $\hat{v}(w) = 10$, $\hat{v}(x) = 12$.

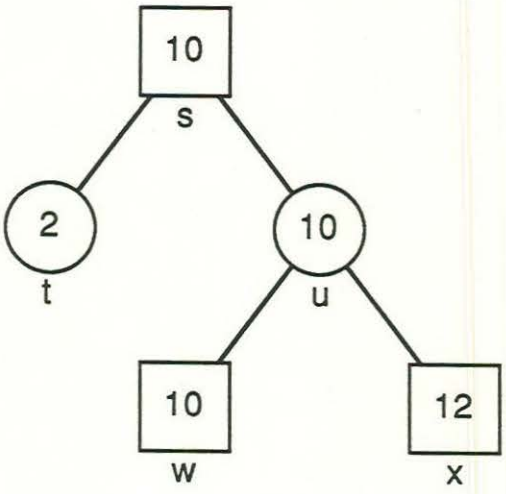

Figure 2: A partial game tree

If we expand at w (say, because that is the most promising line of play), we may obtain Figure 3.

Note that $\hat{v}_E(w)$ has changed from 10 to 11.

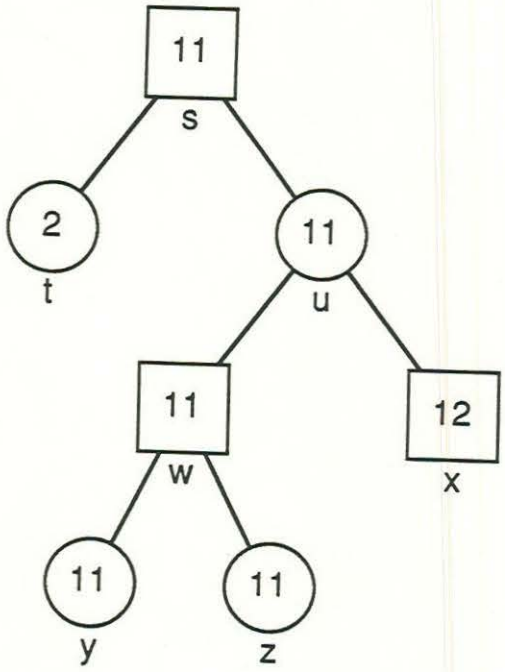

Figure 3: An expanded game tree

Which node should be expanded next? In the conventional framework this question is difficult to answer.

To answer this question we propose picking a relatively large p (e.g., $p = 10$), and computing an approximation $\tilde{v}_E(c)$ to $\hat{v}_E(c)$ for each $c \in E$ by the analog to equation (9) wherein "max" has been approximated by "M_p", and "min" has been approximated by "M_{-p}":

$$\tilde{v}_E(c) = \begin{cases} \hat{v}(c) & \text{if } c \in T(E) \\ M_p(\tilde{v}_E(d_1), \ldots, \tilde{v}_E(d_k)) & \text{if } c \in Max - T(E) \\ M_{-p}(\tilde{v}_E(d_1), \ldots, \tilde{v}_E(d_k)) & \text{if } c \in Min - T(E) \end{cases} \tag{10}$$

where $S(c) = \{d_1, \ldots, d_k\}$. The values $\tilde{v}_E(c)$ should be good approximations to $\hat{v}_E(c)$ if p is sufficiently large.

However, we are less interested in computing the approximate values themselves than we are in computing the derivatives of these values with respect to each argument. In this manner we will be able to derive the "penalties" we need to implement our penalty-based

*We must assume from here on that $\hat{v}(c) > 0$ for all c. This does not affect the structure of the game, since we can add a constant δ to all static values $\hat{v}(c)$'s if necessary. However, it does preclude our using the "negamax" formulation of games [KM75].

scheme. (The penalties will be the negatives of the logarithms of the derivatives – more about this later.)

Let

$$D(x,y) = \frac{\partial \tilde{v}_E(x)}{\partial \tilde{v}_E(y)} \tag{11}$$

where y is any node in E_x, the subtree of E rooted at x. Thus $D(s,c)$ measures the sensitivity of the root value $\tilde{v}_E(s)$ to changes in the tip value $\tilde{v}_E(c)$.

We wish to expand next that expandable tip c with largest value $D(s,c)$. In this manner we may reduce the uncertainty in our estimate $\tilde{v}_E(s)$ in the most efficient manner.

To choose the tip with maximum $D(s,c)$, we can cast our idea as a penalty-based iterative heuristic. We define the weight

$$w(x) = -\log(D(f(x),x)) \tag{12}$$

to be the weight of the edge between $f(x)$ to x.

By the chain rule for differentiation, we have

$$D(s,x) = \prod_{c \in A(x)} D(f(c),c). \tag{13}$$

Since we want to expand the expandable tip x with the *largest* $D(s,x)$, we should choose the expandable tip with the *least* penalty, since the penalties are defined by

$$P_s(x) = \sum_{c \in A(x)} w(c), \tag{14}$$

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

We now redraw Figure 2 as Figure 4, using our 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 $w(d)$. Below each tip c of the tree is written the penalty $P_s(c)$ in brackets. Each such $P_s(c)$ is the sum of the weights $w(d)$ on the edges between c and the root.

According to our rule, w is the best node to expand, since the value at the root depends most strongly on the value at w.

We note that our $\tilde{v}_E$'s and w's satisfy the computability requirements needed for the penalty-based implementation described above.

We now redraw Figure 4 as our new Figure 5, where $\hat{v}_E(c)$ is drawn inside node c as before, and the pair $[a(c), h(c)]$ is placed to the right of node c. Similarly, Figure 6 is our revised version of Figure 5, after w has been expanded. In both figures the edges are labelled with the $w(c)$ values as in Figure 4 for the reader's convenience, although these do *not* need to be explicitly stored.

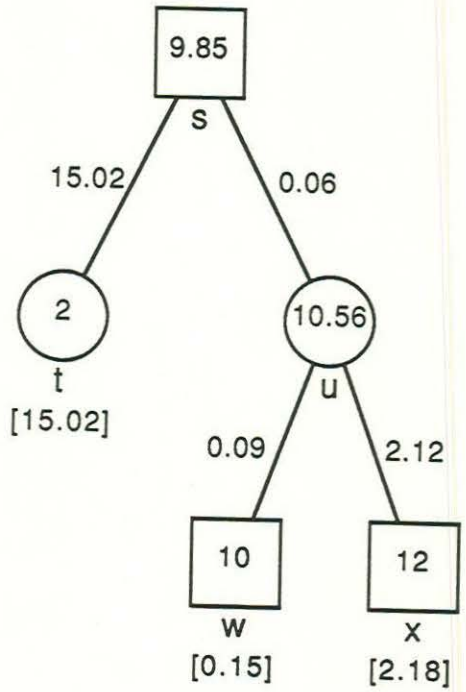

Figure 4:

We see from Figure 6 that tip y is next to be expanded. Here we are still following the most promising line of play for both players. Soon, however, the search will return to expand node x, since $P_s(x) = 1.34$ in Figure 6 is not much larger than $P_s(y) = 1.07$. As the nodes under w are expanded and the tree under w gets larger, the dependence of $\tilde{v}_E(u)$ on any leaf in w's tree will diminish, and eventually x will be selected as the next tip to be expanded.

4 Implementation

If one wishes to experiment with the min/max approximation idea, a major question to be answered is how to cope with the computational difficulty of actually computing the generalized *p-means*.

One can make the argument that one should just "bite the bullet" and compute the generalized means as specified above, in spite of the large computational cost involved in taking powers and roots. This would give the most "accurate" result.

Computing the generalized means exactly might also allow improved play in another

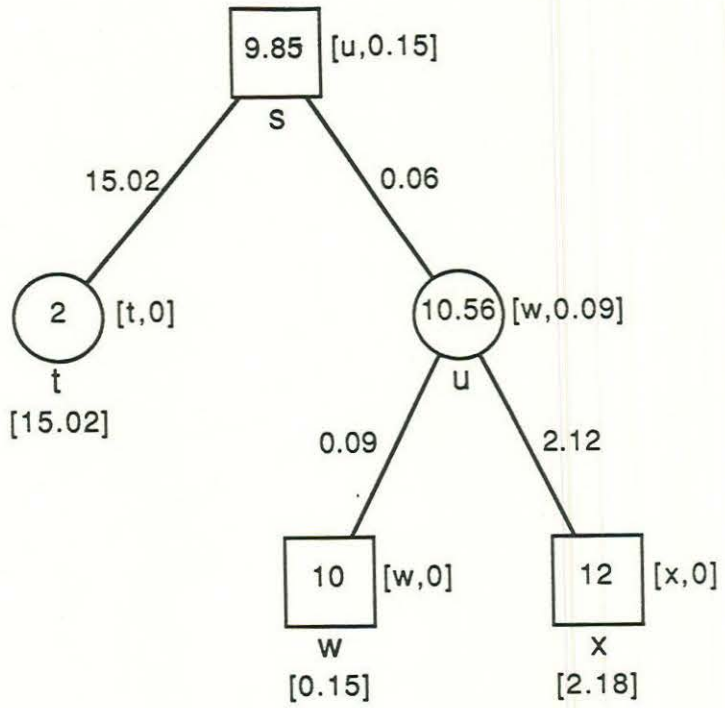

Figure 5:

manner. After the search of the game tree is completed, a move from the root needs to be selected and played. It is most natural to do this by first computing the backed-up estimate $\hat{v}_E(s)$ at the root. However, it may actually be better to use the min/max approximation $\tilde{v}_E(s)$, i.e. selecting the child of the root with maximum $\tilde{v}_E(c)$ instead of the child with maximum $\hat{v}_E(c)$. This has the potential for improved play since the min/max approximation will favor a move whose min/max value can be achieved in several ways over a move whose min/max value can be achieved in only one way. (For example, note that $M_{10}(35,40) > M_{10}(30,40)$ although max $(35,40)$ = max $(30,40)$.) The min/max approximation pays attention to good backup or secondary moves.

Another approach is to skip the computation of the generalized mean values altogether, and use the appropriate min or max values instead. (I.e. use $\hat{v}_E$ instead of $\tilde{v}_E$ everywhere.) Since the generalized mean values are intended to approximate the min and max functions anyway, this may not introduce very much error. The main point of using the generalized mean values was for their derivatives, not for the values themselves. We call this variation the "reverse approximation" idea.

We note that with the "reverse approximation" idea, the weight $w(c)$ is computable from equation (6) as

$$w(c) = \log(n) + (p-1) \cdot (\log(\hat{v}_E(d)) - \log(\hat{v}_E(c))) \tag{15}$$

where c has n siblings, and where d is that sibling of c with the most favorable value $\hat{v}(d)$

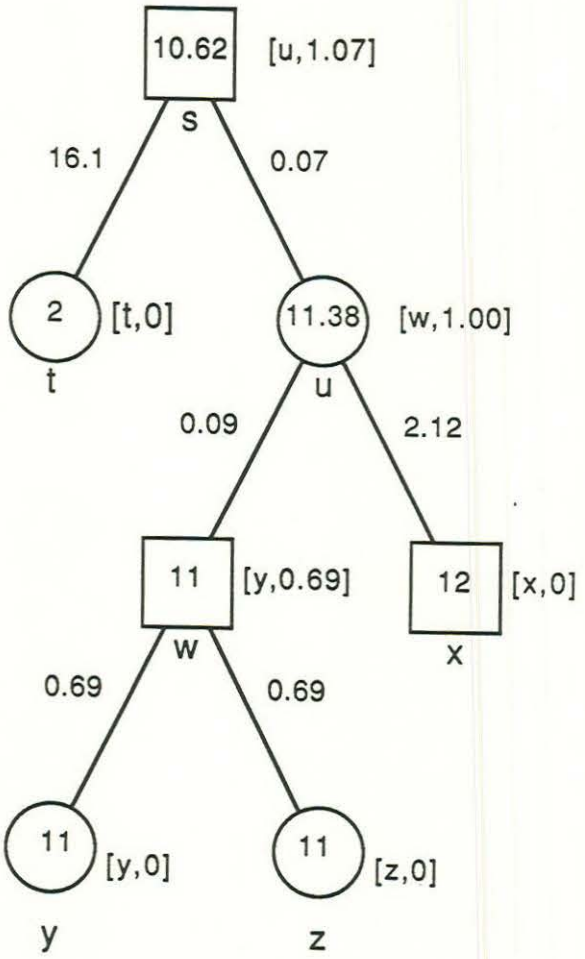

Figure 6:

for the player to move from $f(c)$. Here we might view the $\log(n)$ term as that portion of the penalty associated with descending another level in the tree, and the remaining terms as the penalty associated with making less than optimal moves. If the static evaluation function always returns integers in some range, then each $\hat{v}_E(c)$ is an integer in the same range, and each $w(c)$ can be computed using table-lookups for the logarithm computations.

It is not clear how to "optimally" choose the parameter p. As p gets large, the heuristic should grow very deep but narrow trees. For small p, the heuristic should grow rather broad trees. (For $p = 1$, it would grow the tree in more-or-less of a breadth-first manner, since all derivatives at the same level would be close.) One could conceivably use different p values at different levels of the tree.

5 Empirical Results

We implemented the min/max approximation scheme for the game of "Connect-Four" – a commercially available game, and played it several times against an implementation of the alpha-beta heuristic. Both schemes were given the same static evaluation function to use; this static evaluator returns integer scores in the range 1 to 1023. The alpha-beta heuristic was given a depth-cutoff to limit its search. The min/max heuristic was bounded to use on the average the same number of moves (edge traversals) as alpha-beta had been using up to that point in the game. The min-max heuristic used the "reverse approximation" idea to save time.

We found that min/max heuristic played a solid game, even though it explored many fewer positions on the average than did the alpha-beta heuristic. The games typically ran on until the board was nearly filled up and one of the heuristics was forced to make a losing move (i.e. a move which the other player could play on top of and win immediately). Since the static evaluation did not take into account the number of non-losing moves available to each player, both heuristics suffered from a horizon effect. The games seemed reasonably even overall, with each heuristic winning its fair share.

The min/max approximation heuristic often searched the game tree to twice the depth bound given to the alpha-beta heuristic, and pruned some lines of play quite early (e.g. at depth two).

The completes our description of the initial experiments that have been performed using our new ideas. Of course, the empirical validation of these ideas will require much more extensive testing and optimization than has been performed to date. (We also need to keep in mind, as Nau [Na83] points out, that there are many subtle methodological questions that arise when trying to compare heuristics by playing them off against each other, since the "evenness" of play or variation in quality of play during different portions of the game can have a dramatic influence on the results.)

6 Discussion

We see how our "min/max approximation" heuristic will allocates resources in a sensible manner, searching shallowly in unpromising parts of the tree, and deeper in promising sections. We might also call this approach the "decreasing derivative heuristic", since the nodes are expanded in order of decreasing derivative $D(s, x)$.

It is important to note that the pruning efficiencies exhibited by alpha-beta can also appear with our scheme. Once a move has been refuted (shown to be non-optimal), its weight will increase dramatically, and further exploration down its subtree will be deferred. However, this depends on the static evaluator returning meaningful estimates. If the static

evaluator were to return only constant values except at terminal positions, our scheme would perform a breadth-first search. (We observe that the scheme of McAllester [Mc86] performs like alpha-beta search in this case.)

Other penalty-based schemes are of course possible. We note two in particular:

1. If we can compute an estimate $p(c,d)$ that the actual play would progress from configuration c to successor configuration d, given that play reaches c, then we can define the weight $w(d)$ to be $-\log(p(c,d))$. With this definition, the tip node to be expanded next is the tip node estimated to be *most likely* to be reached in play. This idea was originally proposed by Floyd (see [KM75]), although it does not seem to have been seriously tried.

2. If we estimate for each node c of the probability that c is a forced win for Max (see [Na83] for discussions of this idea) then we can select the tip node to expand upon which our estimate at the root depends most heavily. This can be done, in a manner similar to our min/max approximation technique, beginning with the formulas in [Na83]. This idea was suggested by David McAllester.

7 Open Problems

1. Does our scheme have any advantages in practice, in terms of quality of play?

2. How should one best choose which generalized mean value functions to use?

3. Can our ideas be combined effectively with more traditional approaches? In particular, what is the best way to blend the efficiency of depth-first search with our ideas?

4. How well can these ideas be parallelized?

5. How does the min/max approximation scheme work on pathological games?

8 Conclusion

We have presented a novel approach to game tree searching, based on approximating the min and max functions by suitable generalized mean value functions.

9 Acknowledgements

I would like to thank David McAllester and Charles Leiserson for some stimulating discussions and comments.

10 References

[Be79] Berliner, Hans, "The B* Tree Search Algorithm: A Best–First Proof Procedure," *Artificial Intelligence*, **12**(1979), 23–40.

[BF79] Barr, Avron, and E.A. Feigenbaum (eds.) *The Handbook of Artificial Intelligence* (Kaufmann, 1981) (vol. I), Section II.C.

[CM83] Campbell, Murray S., and T.A. Marsland, "A Comparison of Minimax Tree Search Algorithms," *Artificial Intelligence* **20**(1983), 347–367.

[HLP34] Hardy, G. H., J. E. Littelwood, and G. Polya, *Inequalities* (Cambridge University Press, 1934).

[KM75] Knuth, Donald E., and Ronald M. Moore, "An Analysis of Alpha-Beta Pruning," *Artificial Intelligence*, **6**(1975), 293–325.

[Mc86] McAllester, David A., "A New Procedure for Growing Min-Max Trees," (to appear in *Artificial Intelligence*).

[Na82] Nau, Dana S., "An Investigation of the Causes of Pathology in Games," *Artificial Intelligence* **19**(1982), 257–278.

[Na83] Nau, Dana S., "Pathology on Game Trees Revisited, and an Alternative to Minimaxing," *Artificial Intelligence* **21** (1983), 221-244.

[NPT85] Nau, Dana S., Paul Purdom, and Chun-Hung Tzeng, "An evaluation of two alternatives to minimax," *Proc. on Uncertainty and Probability in Artificial Intelligence* (AAAI, UCLA, August 14–16, 1985), 232–235.

[Ni68] Nilsson, N. J., "Searching Problem-Solving and Game-Playing Trees for Minimal-Cost Solutions," *Proc. IFIP* (1968, vol. 2), 1556–1562.

[Pa85] Palay, A. J., *Searching with Probabilities*, (Pitman, Boston, 1985).

[RP83] Roizen, I., and J. Pearl, "A Minimax Algorithm Better than Alpha-Beta? Yes and No," *Artificial Intelligence* **21** (1983), 199–220.

[St79] Stockman, G., "A Minimax Algorithm Better than Alpha-Beta?," *Artificial Intelligence* **12** (1979), 179–196.