Memo 30 -- Artificial Intelligence Project

RLE and COMPUTATION CENTER
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

December 4, 1961

THE TREE PRUNE (TP) ALGORITHM

by Timothy P. Hart
and
Daniel J. Edwards

Introduction

The Tree Prune (TP) Algorithm is an algorithmic method for pruning unneeded branches from the move tree which is being searched by the standard minimax method. The algorithm makes use of information gained about part of the tree to reject those branches which will not affect the principle variation.

The reasoning behind the TP Algorithm is as follows:

a) if the maximizing-player finds a move whose value is greater than or equal to the value of an alternate minimizing-player move found higher in the tree, he should not look further because the min-player would certainly take that alternate move.

b) if the min-player finds a move whose value is less than or equal to the value of an alternate max-player move found higher in the tree, he should not look further because the max-player would certainly take that alternate move.

Programming Description

The algorithm will be described with reference to a move tree that starts at the top of a page and branches downward (see example). The top of the tree will be called the highest level and the many end points the lowest level.

Rule 1: At any node you reach in the tree keep track of the minimax value of the next higher node.

Rule 2: If the current node is a minimizing node, and a value is found which is less than or equal to the current value of the next higher node, abandon the current node because its value cannot effect the value of the next higher node.

Rule 3: If the current node is a maximizing node, and a value is found which is greater than or equal to the value of the next higher node, abandon the current node because its value cannot effect the value of the next higher node.

Rule 4: During the tree search keep two parameters Alpha and Beta along with their associated levels. Alpha and Beta are initially set to ∞ and $-\infty$ respectively and associated levels to the lowest value corresponding to the deepest you can go in the tree.

When a minimax value for a maximumizing node is found compare the node level with the level associated with alpha. If the node level is equal to or greater than alpha level, set alpha equal to the minimax value and alpha level equal to the node level. If you have set alpha compare alpha level with beta level,and if alpha level is greater than beta level set beta to ∞ and beta level to its initial value.

When a minimax value for a minimizing node is found, compare the node level with the level associated with beta. If the node level is equal to or greater than beta, set beta equal to the minimax value and the beta level equal to the node level. If you have set beta compare the beta level with the alpha level, and if beta level is greater, then set alpha to $-\infty$ and alpha level to its initial value.

Rule 5: Upon reaching a maximizing node for the first time set its initial value equal to alpha.

Rule 6: Upon reaching a minimizing node for the first time set its initial value equal to beta.

LISP Description

The above ideas can be translated into LISP using the following M-expressions. The v^+ and $v1^+$ functions are used for the maximizing player, while the v^- and $v1^-$ are for the minimizing player.

$v^+[p;\alpha;\beta] \equiv [final[p;\alpha;\beta] \rightarrow value[p]; T \rightarrow v1^+[succ[p];\alpha;\beta]]$

$v1^+[list;\alpha;\beta] \equiv [null[list] \rightarrow \alpha; T \rightarrow$
$\lambda[[u];[u > \beta \rightarrow u;$
$T \rightarrow v1^+[cdr[list];max[u;\alpha];\beta]]];$
$v^-[car[list];\alpha;\beta]]$

$v^-[p;\alpha;\beta] \equiv [final[p;\alpha;\beta] \rightarrow value[p];\ T \rightarrow v1^-[succ[p];\alpha;\beta]$

$v1^-[list;\alpha;\beta] \equiv [null[list] \rightarrow \beta;\ T \rightarrow$
$\lambda[[u];[u \leq \alpha \rightarrow u;\ T \rightarrow v1^+[succ[p];\alpha;\beta]]];$
$v^+[car[list];\alpha;\beta]]$

final[p; α;β] is some terminating condition, i.e., depth[p] = nmax.

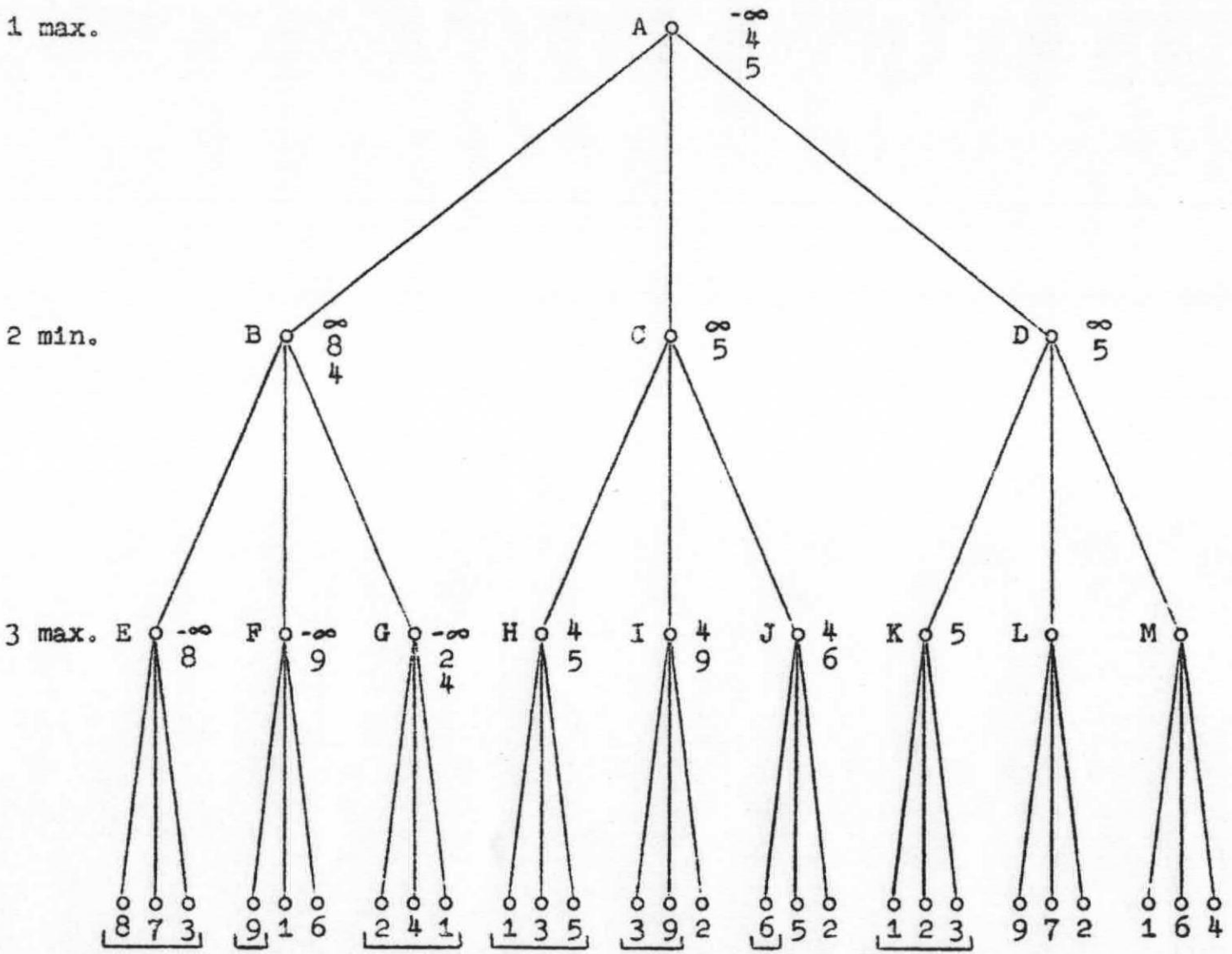
succ[p] gives a list of all legal successor positions from the current position.

Note: This becomes the $\alpha\beta$-Heuristic of Prof. McCarthy if the final function is defined $final[p;\alpha;\beta] \equiv [[opt[p] \leq \alpha] \vee [pess[p] \geq \beta] \vee [depth[p] = nmax]$
where opt[p] is an optimistic value of this position (i.e., what is the most I can hope for from this position) and pess[p] is a pessimistic value (i.e., what is the least I can expect from this position).

Comments

This algorithm preserves the principle variation and has been conservatively estimated to prune from 1/2 to 3/4 of the minimax search tree depending on the final depth to be searched. This algorithm places a premium on heuristically ordering the moves to be tried from any node so that the principle variation may be found quickly and the rest of the tree pruned drastically.

EXAMPLE MOVE TREE



ALPHA		BETA	
Value	Ply	Value	Ply
-∞	99	∞	99
8	3		
∞	99	8	2
9	3		
∞	99	4	2
4	1	∞	99
		5	2
5	1	∞	99
		5	2

└──┘ denotes end points examined.

Values at nodes are in the order of assignment.

Example Description

The tree is investigated from left to right.

Set the value of node A to $-\infty$ from Alpha.

Proceed to B and set to ∞ from Beta.

Proceed to E and set to $-\infty$ from Alpha.

Investigate the first variation and set value of E to 8 and alpha to (8 max, level 3).

Investigate other two variations from E and return to B with value of 8 and set beta to (8 min, level 2).

Proceed to F and set to $-\infty$.

Investigate first variation, get 9 and set the value of F to it. Compare the value with the 8 at B and abandon F as it is greater.

Proceed from B to G and set to $-\infty$.

Investigate variations, finally setting G to 4 and return to B and set beta to (4 min, level 2).

Return to A, set value to 4 and alpha to (4 max, level 1).

Proceed to C and set to ∞ from beta.

Proceed to H and set to 4 from alpha.

Investigate variations and finally set H to 5.

Return to C with 5 and proceed to I.

Set I to 4 from alpha and investigate first two variations. When the 9 is found, it is greater than the 5 at C, so abandon I.

Proceed to J, set it to 4 from alpha. First variation yields 6 which is greater than the 5 at C, so abandon J, return to C, and return to A with value 5.

Set A to 5 and alpha to (5 max, level 1).

Proceed to D, set to ∞ from beta.

Proceed to K and set it to 5 from alpha.

Investigate the variations and return to D with the value still at 5. Since 5 is equal to the value at A abandon D, and the search is over.

Out of the 40 possible positions in the tree, 27 were actually examined, and it is claimed that the ratio of possible positions to searched positions would be even larger for a larger tree.

APPENDIX

It turns out that at best, the TP Algorithm can cut the exponent in the growth rate in half, thus allowing almost twice the search depth. More precisely, we have the following theorem.

Theorem (Levin) : Let n be the number of plies in a tree, and let b be the number of branches at every branch point. Then the number of terminal points on the tree is --

$$T = b^n$$

However, if the best possible advantage is taken of the TP Algorithm, then the number of terminal points that need be examined is --

$$T = b^{\frac{n+1}{2}} + b^{\frac{n-1}{2}} - 1 \qquad \text{for add } n$$

$$T = 2b^{\frac{n}{2}} - 1 \qquad \text{for even } n$$

Proof:

For a convincing personal proof using the new heuristic hand waving technique, see the author of this theorem.