University of California
LAWRENCE RADIATION LABORATORY
Livermore, California

ARTIFICIAL INTELLIGENCE GROUP REPORT NO.3
NOVEMBER 8, 1963

Game Trees, m&n Minimaxing, and the m&n Alpha Beta Procedure

By

James R. Slagle

DOCUMENT NUMBER UCID-4621

ABSTRACT

The author describes what he calls the m&n alpha beta procedure, specifies a LISP computer program for carrying out the procedure and proposes an experiment with it. The proposed experiment may help in eventually obtaining "intelligent" computer programs which can make good decisions based on looking ahead on the "tree" of future possibilities.

JS:cs

1. Overview

1.1 Nature of the m&n Alpha Beta Procedure

m = 1,2,3,...

n = 1,2,3,...

The m&n alpha beta procedure is more efficient than and equivalent to m&n minimaxing (in the same way as the ordinary alpha beta procedure is more efficient than and equivalent to ordinary minimaxing). To obtain the backed-up value to a game position in which it is the maximizing player's turn to move, m&n minimaxing backs up the m best (greatest) values of that position's successors, that is, the positions to which the maximizing player can move. Similarly, to obtain the backed-up value of a position in which it is the minimizing player's turn to move, m&n minimaxing backs up the values of the n best successors. Thus ordinary minimaxing is 1&1 minimaxing, and the ordinary alpha beta procedure is the 1&1 alpha beta procedure.

1.2 The Proposed Experiment

The game of checkers will be used to compare the performance of the 2&2 alpha beta procedure with that of the 1&1 (ordinary) alpha beta procedure. Before this comparison is made, a computer program will "learn" and supply to the 2&2 alpha beta procedure two functions for backing up the values of the two best successors of any position. To do this learning, the program is supplied with a large number of typical checker positions by the experimenter.

2. Purposes of the Proposed Experiment With the 2&2 Alpha Beta Procedure

The proposed experiment may help in eventually obtaining "intelligent" computer programs which make good decisions based on looking ahead on the "tree" of future possibilities. Toward this end, the proposed experiment is directed toward the following, more limited objective. The proposed experiment may help in obtaining game-playing programs which make good moves based on looking ahead on the tree of future, possible positions.

The 1&1 (ordinary) alpha beta procedure is now the best procedure for selectively searching the tree and for backing up values on the tree. The proposed experiment studies an advantage and a disadvantage of 1&1 alpha beta as compared to 2&2 alpha beta. On the one hand, 1&1 alpha beta cuts off its search more readily than does 2&2 alpha beta. On the other hand, a value backed up from the value of only the single best successor of a position is often inferior to a value backed up from the values of the two best successors.

3. Organization of the Report

Section 5., describes 1&1 (ordinary) minimaxing and a weakness eliminated by the m&n alpha beta procedure. Section 6., describes, mainly by example, m&n minimaxing. Section 7., describes, again mainly by example, the equivalent but more efficient m&n alpha beta procedure. Appendix B and Appendix C precisely describe (by LISP[1]computer programs) m&n minimaxing and the m&n alpha beta procedure respectively. Section 8., describes the proposed experiment with the 2&2 alpha beta procedure. Appendix A gives some preliminary definitions needed in Appendix B and Appendix C.

4. Prerequisites

The description of m&n alpha beta is self-contained. However, the reader who is not already familiar with ordinary minimaxing and alpha beta is strongly urged to familiarize himself with them. A brief description of ordinary minimaxing is given in Section 5. A full description of ordinary minimaxing is given by Samuel [2]. Decriptions of ordinary minimaxing and alpha beta are given by Slagle[3].(Everything good in the alpha beta program presented in [3] should be attributed to Professor John McCarthy;

everything bad, to Slagle.) To read the appendixes the reader must be familiar with the notation of LISP[1], a computer language for manipulating symbolic expressions.

5. 1&1 (Ordinary) Minimaxing and a Weakness

The weakness described below of 1&1 minimaxing is shared by the equivalent 1&1 (ordinary) alpha beta procedure. Equivalence means that the move chosen by 1&1 minimaxing using a given termination criterion is always the same as the move chosen by the corresponding 1&1 alpha beta procedure. The m&n alpha beta procedure is designed to eliminate this weakness. In addition, this section establishes some terminology and briefly describes 1&1 minimaxing.

5.1 Brief Description of 1&1 Minimaxing

Assume that the machine has a function, called the evaluation function, which assigns a numerical value to each game position. For definiteness, assume that the greater the value of the function, the better the position tends to be for the machine. For this reason, the machine is called the maximizing player.

In Fig.1, the maximizing player can move from position P to either position P_1 or position P_2. We shall say that the successors of the max-position P are P_1 and P_2. Similarly, the successors of the min-position P_1 are P_{11} and P_{12}. As a convenience to the reader, a horizontal line is drawn on the figures between each min-position and its successors. The machine uses its termination criterion to determine not to search below P_{11}, P_{12}, P_{21}, and P_{22}. Using its evaluation, the machine obtains the values

$$v_{11} = 30 \quad v_{12} = 30 \quad v_{21} = 29 \quad v_{22} = 80$$

for P_{11}, P_{12}, P_{21}, and P_{22}, respectively.

To obtain the backed-up value of a min-position, 1&1 minimaxing backs up the value of the best successor of the min-position. Hence, 1&1 minimaxing backs up 30 to P_1 and 29 to P_2. Hence, 1&1 minimaxing would choose to move to position P_1.

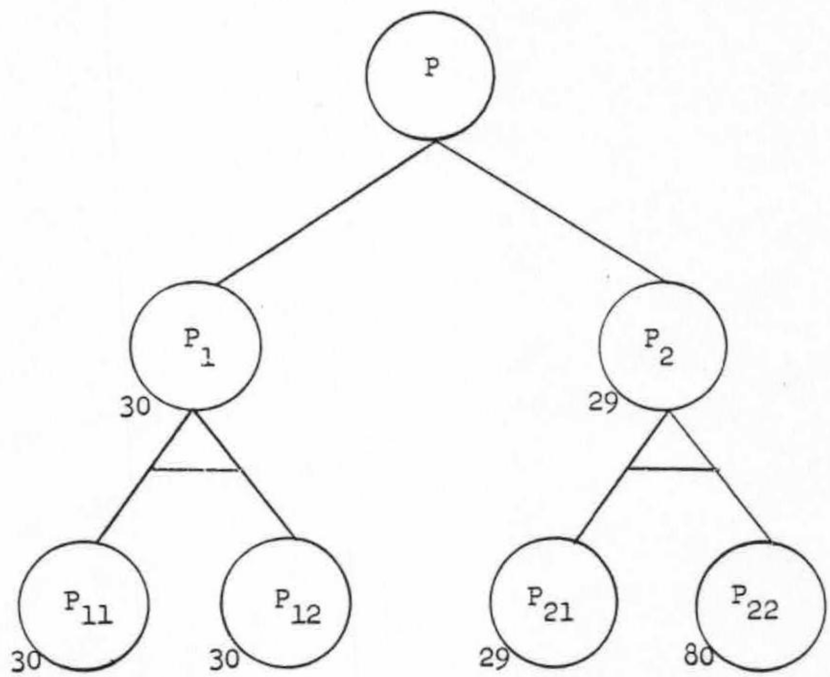


Fig. 1. An Example of Ordinary Minimaxing

5.2 A Weakness of 1&1 Minimaxing

Fig. 1 illustrates a weakness of 1&1 minimaxing. Assume that the maximizing player looking ahead from position P looks at less tree below P_1 than the minimizing player looking from P_1 can look at below P_1. A similar remark applies to P_2. This is generally the case and becomes a near certainty when the number of successors of each position increases to, say, 10 as in checkers. Therefore, the values

$$v_{11} = 30 \quad v_{12} = 30$$

should be considered as the average values which will be found by the minimizing player looking ahead from P_1. A similar remark applies to 29, 80, and P_2. If the uncertainty of these values is sufficiently large, the machine should move to position P_2. The reader should also consider this weakness when 1&1 minimaxing is backing up to a position from its successors deep in a tree.

6. m&n Minimaxing

$$m = 1,2,3,\ldots$$
$$n = 1,2,3,\ldots$$

This section describes m&n minimaxing in order to prepare the reader for the equivalent (but more efficient) m&n alpha beta procedure of the next section. To obtain the backed-up value of a max-position, m&n minimaxing backs up the m best (greatest) values of the successors of the max-position. To obtain the backed-up value of a min-position, m&n minimaxing backs up the values of the n best successors of the min-position. Thus, ordinary minimaxing is 1&1 minimaxing. Appendix B gives a precise description of m&n minimaxing, embodied in a LISP program.

Fig. 2 illustrates 2&2 minimaxing. For the sake of simplicity, assume that the backing up of functions are independent of the depth at which the backing up takes place, although the m&n minimaxing programs in Appendix B make no such assumption. Let the values of the best and second best successors of a max-position be a_2 and α respectively. In our example, we shall assume that the backed-up value to a max-position is given by

$$a_2 + 2^{3-(a_2-\alpha)}$$

Similarly, if the values of the best and second best successors of a min-position are b_2 and β respectively, assume that the backed-up value to a min-position is

$$b_2 - 2^{3-(\beta - b_2)}$$

In Fig. 2, 2&2 minimaxing first backs up the values 15, 20, and 17 to obtain the v_{233}. The values of the n = 2 best successors of the min-position P_{233} are $b_2 = 15$ and $\beta = 17$. Hence, the backed-up value is

$$v_{233} = 15 - 2^{3 - (17 - 15)} = 13$$

Next, 2&2 minimaxing backs up the values 16, 12, and 13. Since the values of the m = 2 best successors of the max-position P_{23} are $a_2 = 16$ and $\alpha = 13$, the backed-up value is

$$v_{23} = 16 + 2^{3 - (16 - 13)} = 17$$

Similarly,

$$v_2 = 13 - 2^{3 - (15 - 13)} = 11$$

Hence, the maximizing player moves to position P_2.

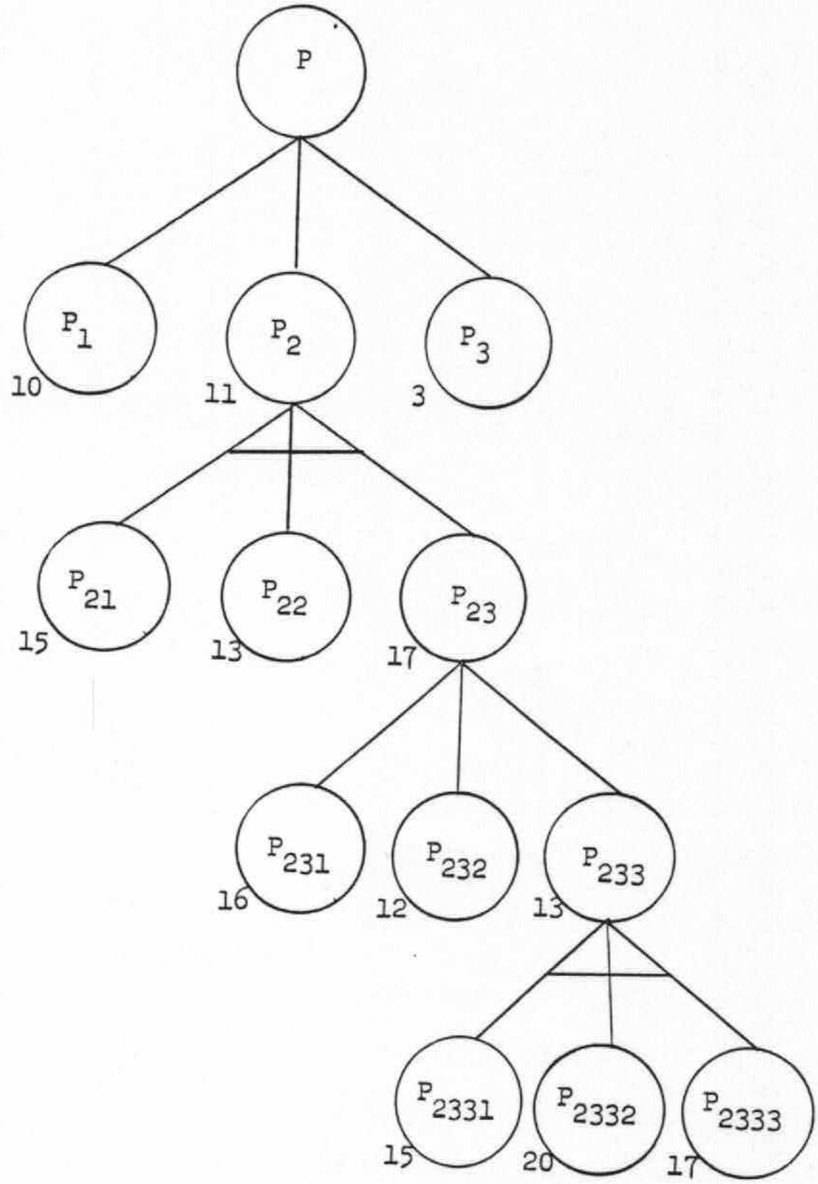


Fig. 2. An Example of 2&2 Minimaxing

7. The m&n Alpha Beta Procedure

$m = 1,2,3,\ldots$

$n = 1,2,3,\ldots$

This section describes the m&n alpha beta procedure, the central idea of this report. The m&n alpha beta procedure is equivalent to m&n minimaxing, that is the move chosen by m&n minimaxing with a given termination criterion is always the same as the move chosen by the corresponding m&n alpha beta procedure. The m&n alpha beta procedure is more efficient than m&n minimaxing, that is m&n minimaxing with a given termination criterion generally looks at much more tree than the corresponding m&n alpha beta procedure does. Ordinary alpha beta is 1&1 alpha beta. Appendix C gives a precise description of m&n alpha beta, embodied in a LISP program with $m = 2,3,4,\ldots$ and $n = 2,3,4,\ldots$ The cases when either m or n is one are excluded only to obtain a slightly more efficient program. The general idea of m&n alpha beta is indicated in the following three examples of increasing interest and complexity.

7.1 A Highest Level m&n Alpha Cutoff

The simplest, although the least interesting, m&n alpha beta cutoff is illustrated in Fig.3. After obtaining $v_1 = 10$, the m&n alpha beta procedure sets $\alpha = 10$ at P_2. After obtaining $v_{21} = 5$, the procedure finds a highest level alpha cutoff, that is the procedure does not bother to look at P_{22} or P_{23} and its successors, but looks next at P_3.

7.2 A 2&2 Beta Cutoff

The 2&2 beta cutoff illustrated in Fig. 4 is a readily anticipated extension of the kind of cutoff which occurs in 1&1 alpha beta. After obtaining $v_1 = 10$, the 2&2 alpha beta procedure sets $\alpha = 10$ at P_2.

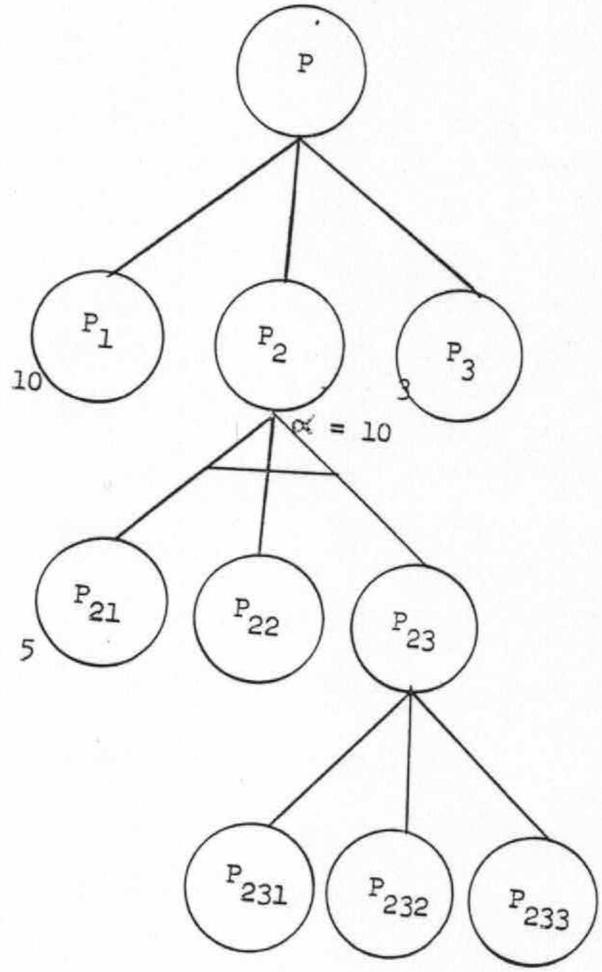


Fig. 3. A Highest Level m&n Alpha Cutoff

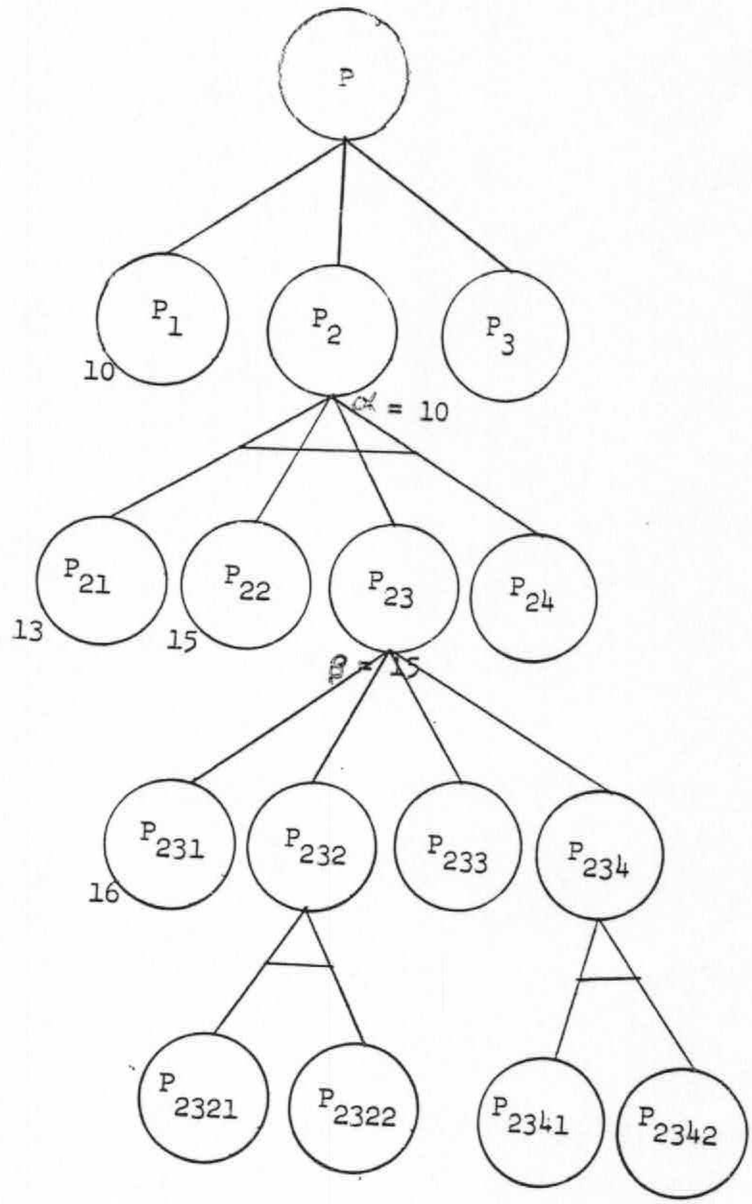


Fig. 4. An Example of 2&2 Beta Cutoff

7.2 (continued

After obtaining $v_{21} = 13$ and $v_{22} = 15$, the procedure sets $\beta = 15$ at P_{23}. After obtaining $v_{231} = 16$, the procedure finds a 2&2 beta cutoff, that is the procedure next looks at P_{24} and never looks at P_{232} (and its successors), P_{233}, and P_{234} (and its successors).

7.3 A 2&2 Alpha Cutoff

Fig. 5 illustrates an interesting, although a somewhat complicated, 2&2 alpha cutoff. Assume that the following function is used to back up to a max-position from its successors, each at depth three. If the values of the best and second best successors of the max-position are a_2 and α respectively, the value to be backed up is

$$a_2 + 2^{3 - (a_2 - \alpha)}$$

On Fig. 5, after obtaining $v_1 = 10$, the 2&2 alpha beta procedure sets $\alpha = 10$ at P_2. After obtaining $v_{21} = 15$ and $v_{22} = 13$, the procedure sets old $\alpha = 10$ (the α of P_2 is the old α of P_{23}) and sets $\beta = 15$ for P_{23}. After obtaining the value $v_{231} = 6$ and $v_{232} = 2$, the procedure sets old $\beta = 15$ and $\alpha = 5$ (not merely 2) for P_{233}. In order to obtain $\alpha = 5$ as the least number which when combined with $v_{231} = 6$ yields at least 10 for the v_{23}, the procedure solves the following equation for α:

$$6 + 2^{3-(6 - \alpha)} = 10$$

After obtaining $v_{2331} = 4$, the procedure finds a 2&2 alpha cutoff, that is the procedure looks next at P_{234} and never looks at P_{2332}, P_{2333}, and their successors. To see that looking at these positions would be a waste of time, first note that $v_{233} \leq 4$.

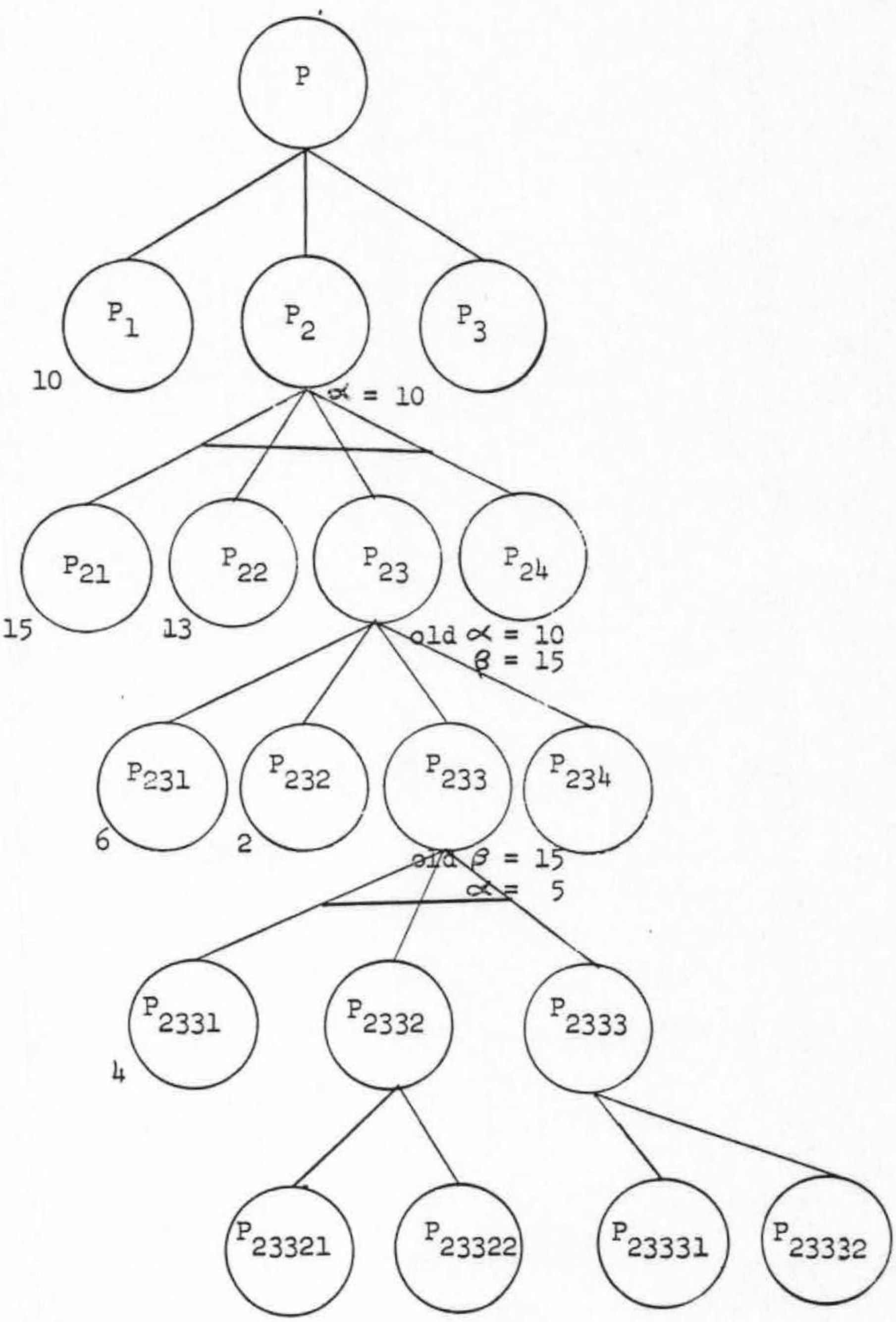


Fig. 5. An Example of 2&2 Alpha Cutoff

7.3 (continued)

There are two cases:

Case I. If P_{233} is not the second best successor of P_{23}, the values of v_{2332} and v_{2333} are irrevelant.

Case II. If P_{233} is the second best successor of P_{23}, the program might just as well use any value less (worse) than v_{233}. In fact, as long as the procedure uses a value v such that

$$v \leq v_{233} \leq 4,$$

a 2&2 alpha cutoff occurs since

$$v_{23} \leq 6 + 2^{3 - (6 - 4)} = 8$$

8. The Proposed Experiment With the 2&2 Alpha Beta Procedure

The game of checkers will be used to compare the performance of 2&2 alpha beta with that of 1&1 alpha beta.

Before this comparison is made, a computer program will "learn" and supply to the 2&2 alpha beta procedure a function for backing up the values of the two best successors of any max-position and the function for backing up the values of the two best successors of any min-position. To do this learning, the program is supplied by the experimentor with a large number of typical checker positions $P^{(k)}$. For definiteness we assume throughout Section 8., that the maximum depth to be searched in a game is 8. First, the program uses 1&1 alpha beta with a maximum depth of 8 to compute a backed up value $V^{(k)}$ for each typical position $P^{(k)}$.

Next the program learns how the backed up value for a position depends on "dmax" for its successors and on the values of the two best successors of the position. The maximum depth to be searched below a position is denoted by dmax, a nonnegative integer, e.g., for each successor of a starting position in a game, dmax is 7. First, the program learns the function for backing up to a min-position from its successors, each having a dmax of 0. Next, this result is used to learn the function for backing up to a max-position from its successors, each having a dmax of 1. Next, both of these results are used to learn the function for backing up a min-position from its successors, each having a dmax of 2, etc. Finally, the first six results are used to learn the function for backing up to a min-position from its successors, each having a dmax of 6.

a. Since dmax = 0 will occur for successors of only min-positions in a game, use procedure 8.2 below to learn the function for backing up the values of the two best successors of a min-position when dmax = 0 for each successor. Use this function in steps b through g below.

b. Since dmax = 1 will occur for successors of only max-positions in a game, use procedure 8.1 below to learn the function for backing up the values of the two best successors of a max-position when dmax = 1 for each successor. Use this function in steps c through g below.

c. Since dmax = 2 will occur for successors of only min-positions in a game, use procedure 8.2 below to learn the function for backing up the values of the two best successors of a min-position when dmax = 2 for each successor. Use this function

8. (continued)

c. (continued)

in steps d through g below.

d. ...

e. ...

f. ...

g. Since dmax = 6 will occur for successors of only min-positions in a game, use procedure 8.2 below to learn the function for backing up the values of the two best successors of a min-position when dmax = 6 for each successor.

8.1 When the Successors Of a Max-position Have a Specified Dmax

For each typical max-position $P^{(i)}$, use 2&2 alpha beta to search to a maximum depth of dmax below each successor of $P^{(i)}$. Let the values of the best and second best successors of $P^{(i)}$ be a_2 and α respectively. Assume the backing up function is of the form

$$a_2 + 2^{x - [a_2 - \alpha]y}$$

where $y > 0$ and x both depend on dmax. This function seems to have roughly the right shape. When $a_2 = \alpha$, the maximum amount, namely 2^x is added to the ordinary minimax value, namely a_2. As $a_2 - \alpha$ increases, the amount added to the ordinary mimimax value decreases asymptotically to zero. The following theorem shows that some care must be taken in the choice of x and y, since $v(a_2)$ should be an increasing function of a_2.

8.1 (continued)

Theorem: Let $v(a_2) = a_2 + 2^{x - (a_2 - \alpha) y}$ (1)

where $y > 0$

The function $v(a_2)$ is increasing on $\{a_2 / a_2 \geq \alpha\}$

if and only if

$$y(2^x) \ln 2 \leq 1 \quad (2)$$

To prove this theorem, considerations with the first and second derivatives of (1) show that $v(a_2)$ has no maximum and only one minimum and that this minimum occurs at

$$a_2 = \alpha + \frac{1}{y} [x + \log_2 (y \ln 2)] \quad (3)$$

Function $v(a_2)$ is increasing on $\{a_2 / a_2 \geq \alpha\}$ if and only if this minimizing $a_2 \leq \alpha$. (4)

Combining (4) and (3) leads to (2).

We now resume our description of how the program learns the function for backing up the values of the two best successors of a max-position. For each typical max-position $P^{(i)}$ the program uses 2&2 alpha beta to search to a maximum depth of dmax below each successor of $P^{(i)}$. Assuming that the backing-up function is of the form

$$a_2 + 2^{x - [a_2 - \alpha] y} \quad (5)$$

where $y > 0$ and x both depend on dmax, the program obtains a collection of data,

$$V^{(i)} = a_2^{(i)} + 2^{x - [a_2^{(i)} - \alpha^{(i)}] y}$$

8.1 (continued)

The program uses some standard approximation technique to find values of x and y which yield a good fit to this data. For this dmax, the program uses this x and this y in (5) for its backing-up function.

8.2 When the Successors of a Min-position Have a Specified Dmax

For each typical min-position $P^{(j)}$, use 2&2 alpha beta to search to a maximum depth of dmax below each successor of $P^{(j)}$. Let the value of the best and second best successor of $P^{(j)}$ be b_2 and β respectively. Assume that the backing-up function is of the form

$$b_2 - 2^{r - [\beta - b_2] s} \qquad (6)$$

where $s > 0$ and r both depend on dmax. The program obtains a collection of data,

$$v^{(j)} = b_2^{(j)} - 2^{r - [\beta^{(j)} - b_2^{(j)}] s} \qquad (7)$$

The program uses some standard approximation technique to find values of r and s which yield a good fit to this data. For this dmax, the program uses this r and this s in (6) for its backing-up function.

Appendix A

TERMINOLOGY AND SOME PRELIMINARY LISP PROGRAMS

Appendix A is intended to prepare the reader, already familiar with LISP [1], for Appendix B and Appendix C. By definition, a final segment of a list $(s_1, s_2, \ldots, s_t)$ is either NIL or $(s_k, s_{k+1}, \ldots, s_t)$. A program which directly uses another program is called a superprogram of that program.

ditto[s;n]

Arguments:

s is any S-expression

n is any nonnegative integer

Value:

a list of n occurrences of s

Example:

ditto[(A . B);2]=((A . B),(A . B))

Superprograms:

sta in both Appendix B and Appendix C

Status:

debugged

Definition:

```
ditto[s;n]=
     cond[zerop[n] → NIL;
          T → cons[s;ditto[s;n-1]] ]
```

insert[sl;el;p]

Arguments:

sl is a sorted list

el is an element to be inserted into the sorted list

p a two-argument predicate defining the ordering

Value:

the sorted list with the element correctly inserted

Example:

insert[(2,6,7);4;LESSP]=(2,4,6,7)

Superprograms:

minvl and maxvl in both Appendix B and Appendix C

Status:

debugged

Definition:

```
insert[sl;el;p]=
        cond[null[sl] → list[el];
            p[el;car[sl]] → cons[el;sl];
            T → cons[car[sl];insert[cdr[sl];el;p]] ]
```

value[p]

Argument:

p is a game position, including whose turn it is to move

Value:

the value (not backed up) of the position

Example:

value[Pl]=10 in Fig.6

Superprograms:

minv and maxv in both Appendix B and Appendix C

Status:

proposed

Definition:

$value[p]=c_1f_1[p]+c_2f_2[p]+c_3f_3[p]$

where c_1, c_2, and c_3 are real(weights) and where f_1, f_2, and f_3 are real-valued functions (features) of the position, for example f_1 might be the piece advantage of black in checkers.

The above is only one of many possible definitions of the function, value.

succ[p] (successors)

Argument:

p is the position

Value:

the successors of p, that is a list of all the positions to which the player whose turn it is can move

Example:

succ[P233]=(P2331,P2332,P2333) in Fig.2

Superprograms:

minv and maxv in both Appendix B and Appendix C

Status:

proposed

Definition:

depends on the particular game and its representation succ[p] (successors)

Argument:

p is the position

Value:

the successors of p, that is a list of all the positions to which the player whose turn it is can move

Example:

succ[P233]=(P2331,P2332,P2333) in Fig.2

Superprograms:

minv and maxv in both Appendix B and Appendix C

Status:

proposed

Definition:

depends on the particular game and its representation

Appendix B

PROGRAMS FOR m&n MINIMAXING

sta[p;m;n;dmax] (start)

Arguments:

p is the starting position, assumed to be a max-position

m=1,2,3,... is the number of values to be backed up to obtain the value of a max-position

n=1,2,3,... is the number of values to be backed up to obtain the value of each min-position

dmax is the maximum depth to be searched below p

Value:

the successor of p which is calculated to be best

Example:

sta[P;2;2;4]=P2 in Fig.2

Superprograms:

none given since sta is the top level m&n minimaxing program

Status:

illustrative example to prepare the reader for the m&n alpha beta programs of Appendix C

Definition:

```
sta[p;m;n;dmax]=
     sta4[p;ditto[-∞;m];ditto[∞;n];dmax]
```

sta4[p;initima;initinb;dmax] (start with 4 arguments)

Arguments:

p is the starting position

initima is the initial list of the m best values on the list ma.

Ordinarily, initima is a list with m members, each

member being $-\infty$

initinb is the initial list of the n best values on the list nb.

Ordinarily, each member of initinb is ∞ .

dmax is the maximum depth to be searched below p

Value:

The best successor of p

Example:

sta4[P;($-\infty$,$-\infty$);(∞,∞);4]=P2

Superprogram:

sta ordinarily. However, sta4 can be used as a top level program

Status:

illustrative

Definition:

sta4[p;initima;initinb;dmax]=sta1[succ[p];$-\infty$;NOSUCC; dmax-1] stal[ℓ;alpha;best;dmax] (starting list)

Arguments:

ℓ is some final segment (initially the entire list) of the list of successors of the starting position

alpha (initially $-\infty$) is the value of the best successor encountered before the final segment

best (initially NOSUCC) is the best successor encountered before the final segment

dmax is the maximum depth to be searched below each successor

Free Variables:

initima and initinb are bound by sta4

Value:

the best successor to the starting position

Example:

stal[(P1,P2,P3);-∞;NOSUCC;3]=P2 in Fig. 2

Superprogram:

sta4

Status:

illustrative

Definition:

```
stal[ℓ;alpha;best;dmax]=
     prog[[u]
          cond[null[ℓ]→ return[best]]
          setq[u;minv[car[ℓ]]]
          return[cond[alpha ≤ u → stal[cdr[ℓ];u;car[ℓ];dmax]
                      T→stal[cdr[ℓ];alpha;best;dmax] ]] ]
```

minv[p] (value of a min-position)

Argument:

p is a min-position

Free Variables:

initima and initinb are bound by sta4

dmax is the maximum depth to be searched below p

Value:

the (generally backed-up)value of p

Example:

minv[P2]=11 in Fig. 2

Superprograms:

stal and mavl

Status:

illustrative

Definition:

```
minv[p]=
     cond[final[p: dmax ]→value[p];
          T→minvl[succ[p];initinb: dmax-1] ]
```

minv[p] (value of a min-position)

Argument:

p is a min-position

Free Variables:

initima and initinb are bound by sta4

dmax is the maximum depth to be searched below p

Value:

the (generally backed-up)value of p

Example:

minv[P2]=11 in Fig. 2

Superprograms:

stal and mavl

Status:

illustrative

Definition:

```
minv[p]=
     cond[final[p; dmax]→value[p];
          T→minvl[succ[p];initinb; dmax-1] ]
```

minvl[ℓ;nb;dmax] (min-position value backed up from the list of its successors)

Arguments:

ℓ is some final segment (initially the entire list) of the list of successors

nb (initially initinb) is a list containing the values of the n best successors of the min-position

dmax is the maximum depth to be searched below each successor

Free Variables:

initima and intininb are bound by sta4

Value:

min-position value obtained by backing up the values of the n best successors of the min-position

Example:

minvl[(P21,P22,P23);(∞,∞);2]=11 in Fig. 2

Superprogram:

minv

Status:

illustrative

Definition:

```
minvl[ℓ;nb;dmax]=
      cond[null[ℓ]→bunb[nb; dmax];
            T→minvl[cdr[ℓ];cdr[insert[nb;maxv[car[ℓ]];GREATERP]];dmax]]
```

maxv[p] (value of a max-position)

Argument:

p is the max-position

Free Variables:

initima and initinb are bound by sta4

dmax is the maximum depth to be searched below p

Value:

value of the max-position

Example:

maxv[P23]=17 in Fig. 2

Superprogram:

minvl

Status:

illustrative

Definition:

```
maxv[p]=
      cond[final[p;dmax]→value[p];
           T→maxvl[succ[p];initima; dmax-1] ]
```

maxvl[ℓ;ma; dmax](value of a max-position as backed up from the list

of its successors)

Arguments:

ℓ is some final segment of (initially the entire list) the list of successors of the max-position

ma(initially initima) is the list of values of the m best successors considered before the final segment

dmax is the maximum depth to be searched below each successor

Free Variables:

initima and initinb are bound by sta4

Value:

value of the max-position obtained by backing up the values of the m best successors of the max-position

Example:

maxvl[(P231,P232,P233);(-∞,-∞);1]=17 in Fig. 2

Superprogram:

maxv

Status:

illustrative

Definition:

```
maxvl[ℓ;ma;dmax]=
     cond[null[ℓ]→buma[ma;dmax];
          T→maxvl[cdr[ℓ];cdr[insert[ma;minv[car[ℓ]];LESSP]];dmax]]
```

final[p;dmax]

Arguments:

p is a position

dmax is the maximum depth to be searched below the position

Free Variables:

possibly some, for example the executive program may bind dmin

Value of the Predicate:

T if and only if no search is to be made below p

Example:

final[Pl]=T in Fig. 2

Superprograms:

minv and maxv

Status:

illustrative

Definition:

final[p;dmax]=zerop[dmax]

The above simple definition is one of many possible definitions. A more complicated definition might use the free variable dmin, as mentioned above.

buma[ma;dmax](back up ma)

Arguments:

ma is the list of values of the m best successors of a max-position

dmax is the maximum depth to be searched below each successor

Value:

the backed-up value of the max-position

Example:

buma[(13,16);1]=17 in Fig. 2

Superprogram:

maxvl

Status:

illustrative

Definition:

$$buma[ma;dmax]=a_2+ 2^{3 - (a_2 - a_1)}$$

where $m = 2$, $a_1 = car[ma]$, and $a_2 = cadr[ma]$

The above is one of many possible definitions.

bunb[nb;dmax](back up nb)

Arguments:

nb is the ordered list of the values of the n best successors of a min-position

dmax is the maximum depth to be searched below each successor

Value:

the backed-up value of the min-position

Example:

bunb[(15,13);2]=11 in Fig. 2

Superprogram:

minvl

Status:

illustrative

Definition:

$$\text{bunb[nb;dmax]} = b_2 - 3 - (b_1 - b_2)$$

where $n = 2$, b_1 = car[nb], and b_2 = cadr[nb]

Appendix C

PROGRAMS FOR THE m&n ALPHA BETA PROCEDURE

m = 2,3,4,...

n = 2,3,4,...

The cases when either m or n is one are excluded only to obtain a slightly more efficient program.

sta[p;m;n;dmax] (start)

Arguments:

p is the starting position, assumed to be a max-position

m is the effective number of values to be backed up to obtain the value of each max-position

n is the effective number of values to be backed up to obtain the value of each min-position

dmax is the maximum depth to be searched below the starting position

Value:

the successor calculated to be best

Example:

sta[P;2;2;5]=P1 in Fig. 6

Superprograms:

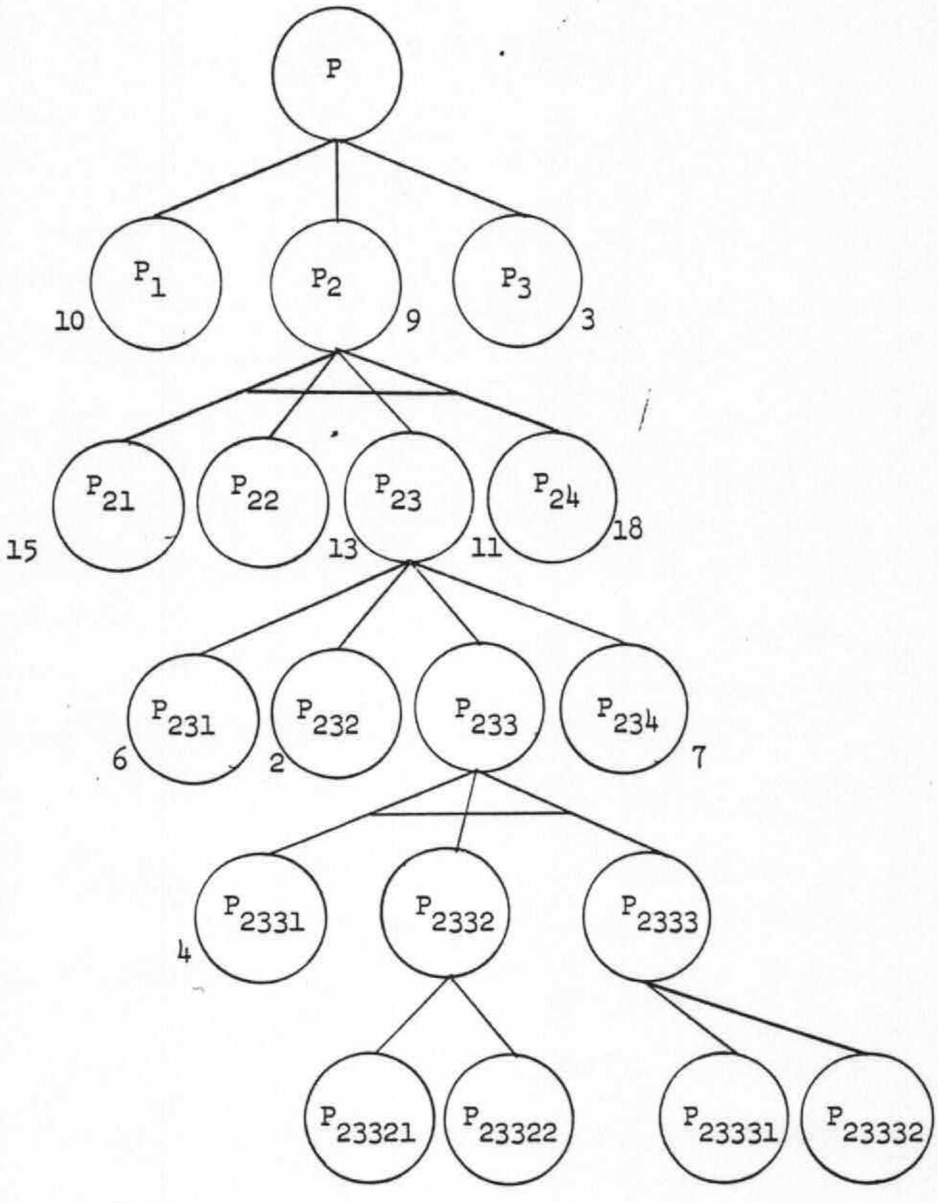
none given since sta is the top level m&n alpha beta program

Status:

proposed

Definition:

sta[p;m;n;dmax]=sta4[p;ditto[- ∞;m - 1];ditto[∞;n - 1];dmax]



Let a_2 = car[a]

The backing up function is bua[α;a;dmax]$=a_2 + 2^{3-(a_2-\alpha)}$

Let b_2 = car[b]

The backing up function is bub[β,b,dmax]$=b_2 - 2^{3-(\beta-b_2)}$

Fig. 6. An Example of 2&2 Alpha Beta.

sta4[p;initia;initib;dmax] (start with 4 arguments)

Arguments:

p is the starting position

initia is the initial value of each list a

Ordinarily each member of initia is $-\infty$

initib is the initial value of each list b

Ordinarily each member of initib is ∞

dmax is the maximum depth to be searched below p

Value:

the successor calculated to be best

Example:

sta4[P;($-\infty$);(∞);5]=P1 in Fig. 6.

Superprogram:

sta ordinarily, although sta4 can be used as a top level program

Status:

proposed

Definition:

sta4[p;initia;initib;dmax]=sta1[succ[p];$-\infty$;NOSUCC;dmax-1] stal[ℓ;alpha;best;dmax] (starting list)

Arguments:

ℓ is some final segment (initially the entire list) of the list of successors of the starting position

alpha (initially $-\infty$) is the value of the best successor before the final segment

best (initially NOSUCC) is the best successor before the final segment

dmax is the maximum depth to be searched below each successor of the starting position

Free Variables:

initia and initib are bound by sta4

Value:

the best successor of the starting position

Example:

stal[(P1,P2,P3);$-\infty$;NOSUCC;4]=P1 in Fig. 6

Superprogram:

sta4

Status:

proposed

Definition:

```
stal[ℓ;alpha;best;dmax]=
      prog[[u];
            cond[null[ ℓ ]→return[best]]
            setq[u;minv[car[ℓ]; ∞]]
            return[cond[alpha ≤ u→stal[cdr[ ℓ ];u;car[ ℓ ];dmax];
                             T→stal[cdr[ ℓ ];alpha;best;dmax] ]] ]
```

minv[p;oldbeta] (value of a min-position)

Arguments:

p is the min-position

oldbeta is the beta of the predecessor of the min-position

Free Variables:

initia and initib are bound by sta4

alpha is bound by stal,maxvl,and maxvl3. See also minvl and minvl3

dmax is the maximum depth to be searched below the min-position

Value:

the (generally backed up)value of p

Example:

minv[P233;15]=4 in Fig. 6

Superprograms:

stal,maxvl,and maxvl3

Status:

proposed

Definition:

```
minv[p;oldbeta]=
     cond[final[p;alpha;oldbeta;dmax]→value[p];
          T→minvl[succ[p];∞;initib;dmax-1] ]
```

minvl[ℓ ;beta;b;dmax] (min-position value as backed up from the list of its successors

Arguments:

ℓ is some final segment (initially the entire list) of the list of successors of the min-position

beta is the same as car[b], that is the value of the $(n - 1)^{th}$ best successor considered before the final segment

b (initially initib) is the list of values of the n - 1 best successors considered before the final segment

dmax is the maximum depth to be searched below each successor

Free Variables:

initia and initib are bound by sta4

alpha When alpha $\geq$ the value of some member of ℓ, an m&n alpha cutoff occurs. See also the binding functions stal, maxvl, and maxvl3.

oldbeta is the beta of the predecessor of the min-position

beta = car[b] > lubub[oldbeta;b;dmax]

Value:

min-position value obtained by backing up the values of the n best successors of the min-position

Example:

minvl[(P21,P22,P23,P24);∞;(∞);3]=9 in Fig. 6.

Superprogram:

minv

Status:

proposed

Definition:

```
minvl[ℓ;beta;b;dmax]=
    prog[[u;newb;lubub];
        cond[null[ℓ]→return[bub[beta;b;dmax]]];
        setq[u;maxv[car[ℓ];alpha]]
        cond[alpha≥u→return[u];
            u≥beta→return[minvl[cdr[ℓ];beta;b;dmax]] ];
        setq[newb;insert[cdr[b];u;GREATERF]];
        setq[lubub;lubub[oldbeta;newb;dmax]];
        return[cond[lubub≥car[newb]→minvl3[cdr[ℓ ]·
                                              min[beta;lubub];
                                              newb];
                  T→minvl[cdr[ℓ];car[newb];newb;dmax] ]] ]
```

minvl3[ℓ;beta;b] (value of a min-position as backed up from the list of its successors, with 3 arguments)

Arguments:

ℓ is some final segment (never the entire list) of the list of successors of the min-position

beta is min[b_1;lubub[oldbeta;b;dmax]] where b_1 is the value of the n^{th} best successor considered before the final segment

b is the ordered list of values of the n - 1 best successors considered before the final segment

Free Variables:

initia and initib are bound by sta4

alpha is bound by stal, maxvl, and maxvl3. When alpha ≥ the value of some member of the final segment, an m&n alpha cutoff occurs.

dmax is the maximum depth to be searched below each member of ℓ

Value:

min-position value obtained effectively by backing up the values of the n best successors of the min-position

Example:

minvl3[(P22,P23,P24);∞;(15)]=9 in Fig. 6.

Superprogram:

minvl

Status:

proposed

Definition:

```
minvl3[ℓ;beta;b]=
     prog[[u];
          cond[null[ℓ]→return[bub[beta;b;dmax]]]
          setq[u;maxv[car[ℓ];alpha]];
          return[cond[alpha≥ u → u;
                         u≥ car[b]→minvl3[cdr[ℓ];min[u;beta];b];
                         T→minvl3[cdr[ℓ];car[b];insert[cdr[b];u;GREATERP]] ]] ]
```

maxv[p;oldalpha] (value of a max-position)

Arguments:

p is the maximum position

oldalpha is the alpha of the predecessor of the max-position

Free Variables:

initia and initib are bound by sta4

beta is bound by minvl and minvl3. See also maxvl and maxvl3

dmax is the maximum depth to be searched below the max-position

Value:

the (generally backed-up)value of p

Example:

maxv[P23;10]=11 in Fig. 6.

Superprograms:

minvl and minvl3

Status:

proposed

Definition:

```
maxv[p;oldalpha]=
     cond[final[p;oldalpha;beta; dmax]→value[p];
          T→maxvl[succ[p];-∞;initia;dmax-1] ]
```

maxvl[ℓ ;alpha;a;dmax] (value of the max-position as backed up from the list of its successors]

Arguments:

ℓ is some final segment (initially the entire list) of the list of successors of the max-position

alpha is the same as car[a], that is the value of the $(m - 1)^{th}$ best successor considered before the final segment

a (initially initia) is the list of values of the m - 1 best successors considered before the final segment

dmax is the maximum depth to be searched below each successor

Free Variables:

initia and initib are bound by sta4

beta is bound by minvl and minvl3. When beta $\leq$ the value of some member of the final segment, an m&n beta cutoff occurs

oldalpha is the alpha of the predecessor of the max-position

alpha = car[a] $<$ glbua[oldalpha;a;dmax]

Value:

max-position value obtained effectively by backing up the values of the m best successors of the max-position

Example:

maxvl[(P231,P232,P233,P234);$-\infty$;($-\infty$);2]=11 in Fig.6.

Superprogram:

maxv

Status:

proposed

Definition:

```
maxvl[ℓ;alpha;a;dmax]=
        prog[[u;newa;glbua];
            cond[null[ℓ]→return[bua[alpha;a;dmax]]]
            setq[u;minv[car[ℓ];beta]];
            cond[beta≤u→return[u];
                u≤alpha→return[maxvl[cdr[ℓ];alpha;a;dmax]] ];
            setq[newa;insert[cdr[a];u;LESSP]];
            setq[glbua;glbua[oldalpha;newa;dmax]];
            return[cond[glbua≤ car[newa]→maxvl3[cdr[ℓ];
                                                max[alpha;glbua];
                                                newa];
                    T→maxvl[cdr[ℓ];car[newa];newa;dmax] ]] ]
```

maxvl3[ℓ;alpha;a] (value of a max-position as backed up from the list of its successors, with 3 arguments)

Arguments:

ℓ is some final segment (never the entire list) of the list of successors of the max-position

alpha is max[a_1;glbua[oldalpha;a;dmax]] where a_1 is the value of the m^{th} best successor considered before the final segment

a is the ordered list of values of the m - 1 best successors considered before the final segment

Free Variables:

initia and initib are bound by sta4

beta is bound by minvl and minvl3. When beta ≤ the value of some member of the final segment, an m&n beta cutoff occurs

dmax is the maximum depth to be searched below each member of ℓ

Value:

max-position value obtained effectively by backing up the values of the m best successors of the max-position

Example:

maxvl3[(P232,P233,P234);5;(6)]=11 in Fig. 6

Superprogram:

maxvl

Status:

proposed

Definition:

```
maxvl3[ℓ;alpha;a]=
     prog[[u];
          cond[null[ℓ]→return[bua[alpha;a;dmax]]];
          setq[u;minv[car[ℓ];beta]];
          return[cond[beta≤ u→u;
                      u ≤ car[a]→maxvl3[cdr[ℓ];max[u;alpha];a];
                      T→maxvl3[cdr[ℓ];car[a];insert[cdr[a];u;LESSP]] ]] ]
```

final[p;alpha;beta;dmax]

Arguments:

p is a position

alpha See the description of the superprogram

beta See the description of the superprogram

dmax is the maximum depth to be searched below the position

Free Variables:

possibly some, for example dmin might be bound by some executive program

Value of the Predicate:

T if and only if no search is to be made below p

Example:

final[P22];10;∞;3]=T in Fig.6.

Superprograms:

minv and maxv

Status:

proposed

Definition:

final[p;alpha;beta;dmax]=zerop[dmax]

The above simple definition is one of many possible definitions. A more complicated definition might use the free variable, dmin, mentioned above.

bua[alpha;a;dmax] (back up a)

Arguments:

alpha is as in the superprogram and is effectively the value of the m^{th} best successor of the max-position

a is the ordered list of values of the m - 1 best successors of the max-position

dmax is the maximum depth to be searched below each successor

Value:

the backed-up value of the max-position

Example:

bua[6;(7);2]=11 in Fig 6

Superprograms:

maxvl and maxvl3

Status:

proposed

Definition:

$$\text{bua[alpha;a;dmax]} = a_2 + 2^{x - (a_2 - \alpha)y}$$

where $m = 2$, a_2 = car[a], and both $y > 0$ and x depend on dmax glbua[oldalpha;a;dmax] (greatest lower bound for backing up a)

Arguments:

oldalpha is alpha for the predecessor of a max-position

a is the ordered list of values of the m - 1 best successors of the max-position

dmax is the maximum depth to be searched below each successor

Value:

the least value which, combined with the values on a, yields a backed-up value of at least oldalpha

Example:

glbua[10;(6);2]=5 in Fig 6.

Superprogram:

maxvl

Status:

proposed

Definition:

glbua[oldalpha;a;dmax]=

cond[oldalpha $\leq a_2 \rightarrow -\infty$;

$T \rightarrow a_2 + \frac{1}{y}$ [$\log_2$ [oldalpha $-a_2$] -x]]

The expression to the right of the second arrow is obtained by solving the following equation for alpha:

$$\text{oldalpha} = a_2 + 2^{x - (a_2 - \alpha)y}$$

The above definition of glbua is one of many possible definitions. The definition is derived from the definition of bua. For example, the above definition is derived from the sample definition given in the description of bua, namely

$$\text{bua[alpha;a;dmax]} = a_2 + 2^{x - (a_2 - \alpha)y}$$

bub[beta;b;dmax] (back up b)

Arguments:

beta is as in the superprograms and is effectively the value of the n^{th} best successor of a min-position

b is the ordered list of values of the n - 1 best successors of the min-position

dmax is the maximum depth to be searched below each successor

Value:

the backed-up value of the min-position

Example:

bub[13;(11);3]=9 in Fig 6.

Superprograms:

minvl and minvl3

Status:

proposed

Definition:

$$\text{bub[beta;b;dmax]} = b_2 - 2^{r - (\beta - b_2)s}$$

where n = 2. Both $s > 0$ and r depend on dmax. b_2 = car[b]

The above is one of many possible definitions.

lubub[oldbeta;b;dmax] (least upper bound for backing up b)

Arguments:

oldbeta is beta for the predecessor of a min-position

b is the ordered list of values of the n - 1 best successors of the min-position

dmax is the maximum depth to be searched below each successor

Value:

the greatest value which, when combined with the values on b, yields a backed-up value of at most oldbeta

Example:

lubub[∞;(15);3]= ∞ in Fig 6.

Superprogram:

minvl

Status:

proposed

Definition:

lubub[oldbeta;b;dmax]=

cond[oldbeta $\geq b_2 \rightarrow \infty$;

$T \rightarrow b_2 + \frac{1}{s}$ [r - $\log_2$ [b_2 - oldbeta]]]

The expression to the right of the second arrow is obtained by solving the following equation for beta:

$$\text{oldbeta} = b_2 - 2^{r - (\beta - b_2)s}$$

The above definition of lubub is one of many possible definitions. The definition is derived from the definition of bub. For example, the above definition is derived from the sample definition given in the description of bub, namely

$$\text{bub[beta;b;dmax]} = b_2 - 2^{r - (\beta - b_2)s}$$

JS:cs

REFERENCES

1. McCarthy, John, et al, Lisp 1.5 Programmers Manual, MIT Computation Center and RLE, August 17, 1962.

2. Samuel, Arthur, "Some Studies In Machine Learning Using the Game of Checkers", IBM Journal of Research and Development, Vol. III, 3, July, 1959, p. 210-229.

3. Slagle, James, Mathematical Theory of Computation and Artificial Intelligence (6.539 class notes), MIT, 1963.

JS:cs