Alpha-Beta Pruning

W D Maurer
George Washington University
SEAS
Washington DC 20052

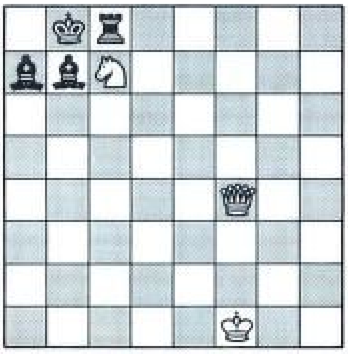
Get your shears out, and get ready to cut back your game trees, thereby saving both space and time

Sooner or later, almost everyone with a small system gets the idea of programming it to play chess, checkers, or some other two-person board game. Most of us give up before we start because we have no idea how to determine the best move in any given situation. The other aspects of playing a game are generally no problem.

We can see how to represent 64 squares on a board by 64 bytes in memory, each of which contains a code number which might be 3 for Bishop, 6 for King, or 0 for a blank square, and so on. We can see how to write a program for each piece, determining where it can move in a given situation depending upon the rules of the game. For example, a Bishop can move as far as possible in any of four directions, so we have to write a program to search in one direction until it finds a square that is not blank (ie: the corresponding byte does not contain 0, the code for a blank square). If this square is *n* squares away from where the Bishop is currently positioned, then there are $n - 1$ possible moves that the Bishop can make in that direction. This loop is then repeated, once for each of the four directions.

Finally, we can see how to write a program that would find all of the pieces on the board, would determine the type of each piece, and would find all possible moves for each piece, according to its type. In this way we could get a list of all of the moves that could be made by one player in any given situation. But to find the best of these defies the low-level intuition that most of us rely upon.

In this article, I will describe a general procedure for programming board games, relying heavily on chess in my examples, but utilizing procedures that can be applied in any board game where you have to "look ahead." The logic is roughly as follows: if I make move X, then my opponent will make move Y; if I make move Z, then my opponent can make move U, which is better for him than move Y, so I shouldn't make move Z; but if I make move W...and so on.



```
1. N-R6 dbl ch     K-R1
2. Q-N7 ch         R x Q
3. N- B7 mate
```

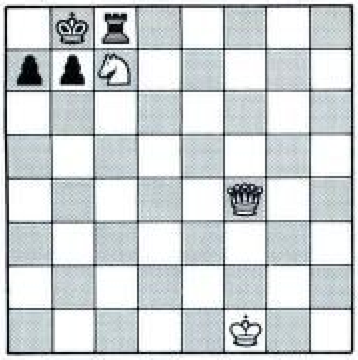
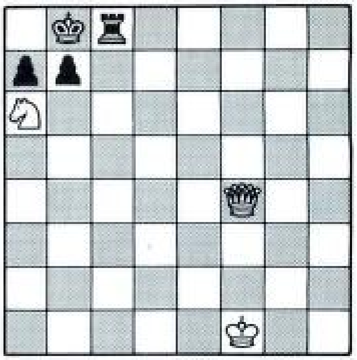
Figure 1: *Chessboard layout just prior to the conclusion of a famous dramatic ending to a chess game.*

The first illustration will be from a famous dramatic finish to a chess game. This is illustrated in figure 1. White is already far ahead, having a Queen and a Knight, whereas Black has only a Rook and two pawns. To finish the game quickly, White lets Black capture his Queen, then gives checkmate with his Knight. For those who have forgotten their chess (and also to illustrate what the computer does when it sees this position), the entire finish of the game is illustrated in figure 2 (see page 88).

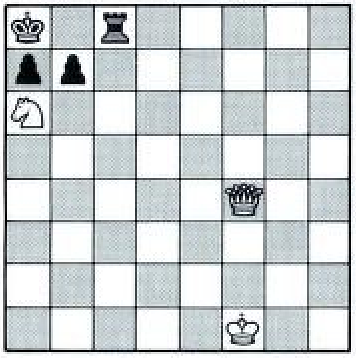
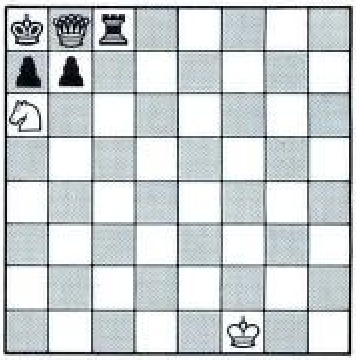
It is clear that the computer has to perform a complete analysis of the given position in a game; much more complete than that given in either figure 1 or figure 2. For example, look at White's first move: N-R6 double check. In chess terminology, as soon as White makes this move, Black's next move is "forced." There is nothing that Black can do except move K-R1. But what does this mean? Black actually has several moves, but all of the others are illegal because White would be able to capture his King. Specifically:

- If Black plays R-B2 (interposing the Rook), then White plays NxK (capturing the King with his Knight).
- If Black plays PxN (capturing the Knight), then White plays QxK

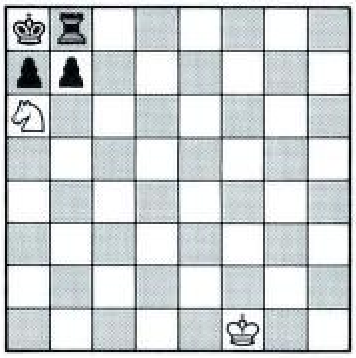
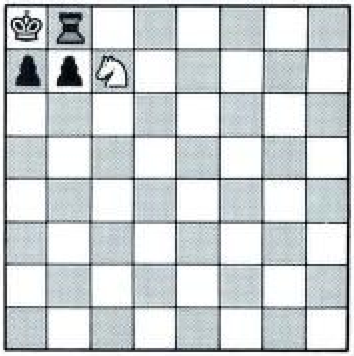
Text continued on page 90

IT IS WHITE'S TURN TO MOVE, AND..............WHITE CHECKS WITH BOTH QUEEN AND KNIGHT. BLACK IS FORCED........

....TO MOVE INTO THE CORNER, AND...........NOW WHITE SACRIFICES THE QUEEN.

THERE IS NOTHING THAT BLACK CAN DO BUT TO TAKE THE QUEEN.........

.....WHEREUPON WHITE GIVES CHECKMATE.

Figure 2: *The sequence of moves that White makes to capture Black's King . . . . CHECKMATE!*

Text continued:

(capturing the King with his Queen).

- If Black plays anything else, then White can play either NxK or QxK.

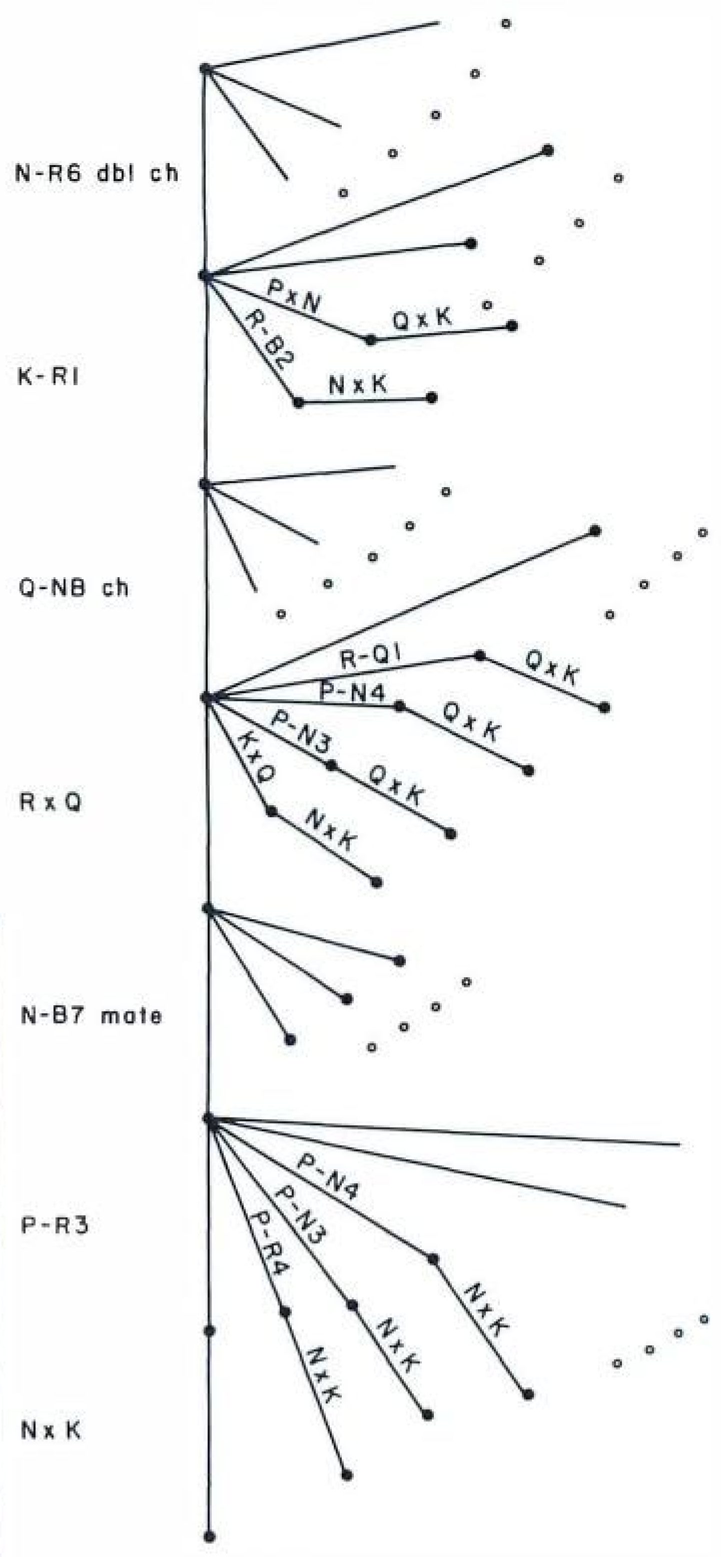
You might argue that the computer does not need to perform all of this analysis, because there is an old rule that states when you are in double check, you have to move your King—there is no other way out. This is perfectly true, but how do you know that you are in double check in the first place, without a similar analysis? It is easier to run through all of the moves, as described above, and verify that, in every case but one, Black's King would be captured. Additionally, look at the next position. Black does play K-R1, and now White plays Q-N8 check. This time Black is not in double check, but his next move is still forced, and Black's King can be captured in two different ways if he does not make the move he is forced to make. Specifically:

- If Black plays KxQ (capturing with the King instead of with the Rook), then White plays NxK.
- If Black plays P-N3 (or any other move than RxQ or KxQ), then White plays QxK.

When Black plays RxQ, White plays N-B7, which is checkmate. But the computer's job is still not finished. How can you tell that this is checkmate? The only way to tell is to look at all of Black's possible moves and make sure that White can capture Black's King in each case. From the computer's point of view, the game is never over until the King is actually captured.

A diagram of the analyses that have been carried out so far would look like figure 3. Each point (dot) in this figure denotes a position of the board. The lines between board positions denote moves. The actual moves that have been made are at the left, but there are other moves which were not taken. In Black's case, each of these led to Black's King being captured. In White's case, they were simply other possible moves that were not made because White has a way, as shown, of winning the game. This diagram is called a *game tree*.

Figure 3: *An illustration of the game tree diagram. A complete game tree diagram would enumerate all possible moves so that the optimum move could be chosen.*



The game tree of figure 3 is a bit hard to visualize because there are so many possible moves. Therefore, in order to illustrate the processing of game trees by computer, I have drawn a simplified game tree in figure 4. In this game tree there are only two possible moves for White at each point, and only two possible moves for Black. This will almost never be the case in a real game situation; here it allows the tree to fit easily on one piece of paper, so that it can be readily visualized. Like any tree, this tree has leaves, branches, and a root; in this case A, B, C...through P are the leaves, 5 is the root, and all of the other nodes are branches.

YOUR PERSONAL

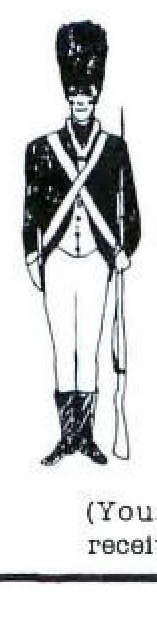
COMPUTER PROGRAM

Could Be Worth $50 to $100,000 In Royalty Payments

There is a big demand for Personal Computer Programs of all categories: Games, Entertainment, Educational, Personal Finance, Hobbies, Diet Nutrition, Small Business, Doctors, Lawyers, Dentists, Homeowners, etc.

You can give the programs you worked so hard to create "Maximum Exposure" with no risk of plagiarism by registering with PROTECT Registry and Referral Service.

- We will send your program description to 25 major marketers — all inquiries will be sent to you for direct negotiation.
- We will furnish copyright application materials with simple instructions, and also send in your copyright application.
- We will register your program and keep it on file for you.



GET STARTED TODAY

SEND $5.00 (Cash - Check - Money order) to:

PROTECT, INC.

P.O. BOX 502

FRANKLIN PARK, IL. 60131

We will send you registry and copyright materials, major marketer list and details of our service.

(Your money cheerfully refunded if you are not satisfied with what you receive - Just Send Back ALL Materials Within 30 Days For Full Refund).

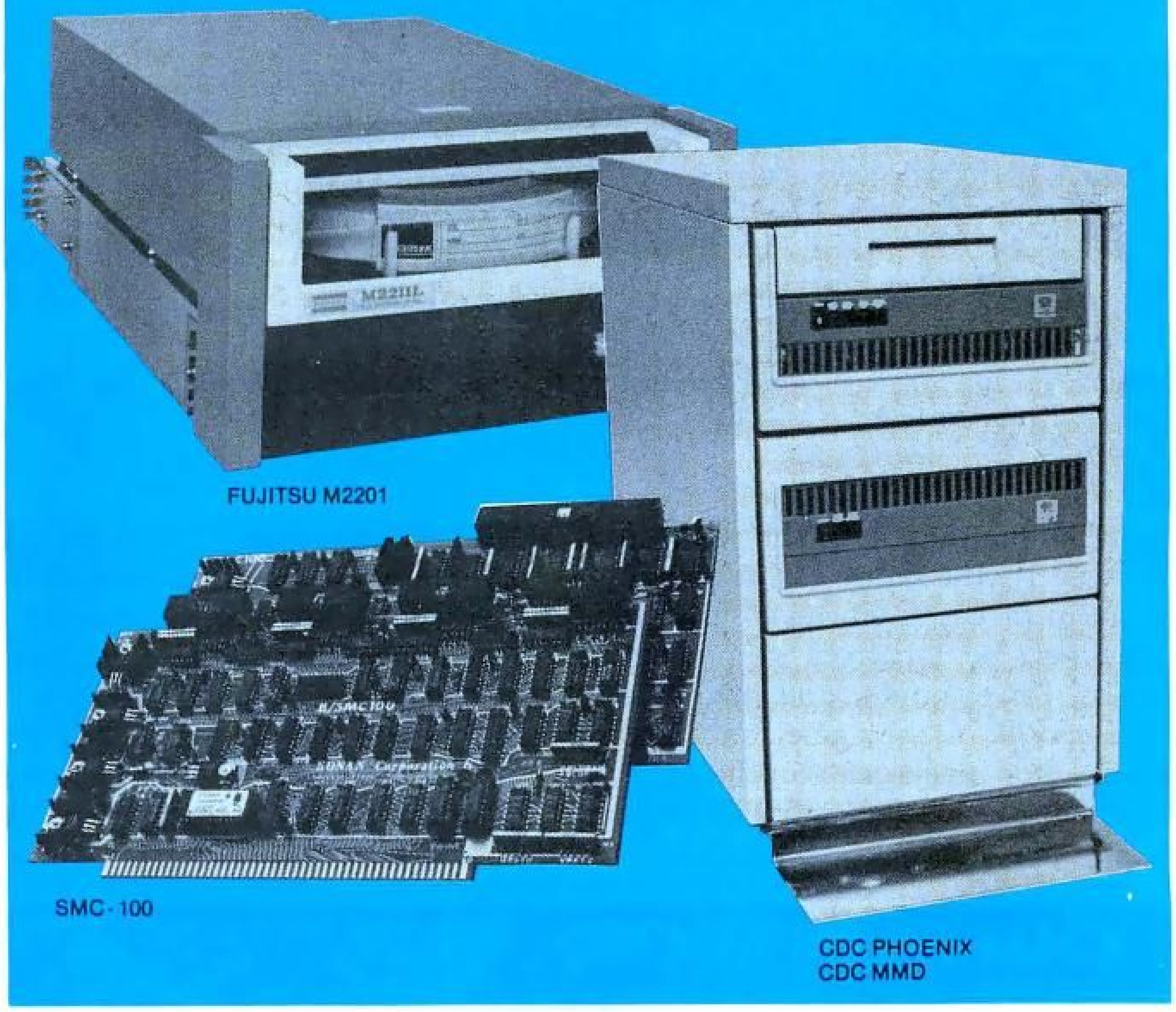
Circle 336 on inquiry card.

In any game tree, the first question you must ask is whether or not it is complete. A game tree is complete if every one of its leaves corresponds to the end of the game. In figure 3, all leaves that are shown correspond to the end of the game (the King is captured), but there are some other leaves, not shown, that do not have this property. If a game tree is complete, it should be obvious that we can tell who ought to win, and the winning strategies. Suppose that the leaves B, L, A, C, and K represent a win for Black, and all other leaves represent a win for White. White (moving first) can win by moving to branch 4. Black will move to branch 1, and White now moves to branch U, winning regardless of Black's move (moving to leaf I or J).

Furthermore, this is the only winning strategy for White. If White's first move is to branch 3, then Black moves to branch Y, and Black now wins, no matter what White does (moving to branch Q or R). If White moves to branch V on his second move, then Black wins by moving to either K or L. This state of affairs will not always hold. There are positions in which White can win no matter what his first move is (suppose, for example, Black's winning positions were B, L, A, E, K...figure it out for yourself). There are also positions in which White cannot win, no matter what his first move is. If Black's winning positions are B, L, I, C, and K, and White starts by moving to 3, then Black moves to Y, whereas if White starts by moving to 4, Black moves to 1. In either case, Black can eventually win.

Now suppose that the game tree is not complete. This is presumably because it is so large that you would run out of memory if you tried to store the complete tree, so you would only store part of it. In this case it is still quite possible that there is a winning strategy for one player or the other. Suppose that Black's winning positions are B, L, I, C, and K, as in the last of the three examples above, but the other leaves of the tree are not winning positions for either White or Black. (In fact, these are not really leaves; if I had room to keep more of this game tree, I could consider further moves beyond each of these points.) It is clear that Black can still

UP TO 2400 MEGABYTES OF HARD DISK CONTROL FOR THE S-100 BUS



Konan's SMC-100 is versatile, fast, cost efficient. It's the disk controller that brings S-100 bus micro computers together with large capacity hard disk drives.

Versatile

Interfaces S-100 bus micro computerswith all fixed or removable media disk drives with storage module (SMD) interfaces. Each Konan SMC-100 will control up to 4 drives ranging from 8 to 600 megabytes per drive, including most "Winchester" type drives. Up to 2400 megabytes of hard disk per controller! And you can take your pick of hard disk drives: Kennedy, Control Data, Fujitsu, Calcomp, Microdata, Memorex, and Ampex, for example.

Fast

SMC-100 transfers data at fast, 6 to 10 megahertz rates, with full onboard sector buffering and sector interleaving, and a DMA that's faster than other popular S-100 DMA controllers.

Cost efficient

SMC-100 is priced right to keep your micro computer system micro-priced. It takes advantage of low-cost-per-megabyte disk drive technology to make the typical cost less than $80 per megabyte.
The OEM / Dealer single quantity price is only $1650, with driver ROM option. Excellent quantity discounts are available.

SMC-100 availability:

Off the shelf to 30 days in small quantities. (Complete subsystems are on hand for immediated delivery.)

Konan has the answers. Talk to them today. Call direct on Konan's order number: 602-269-2649. Or write to Konan Corporation, 1448 N. 27th Avenue, Phoenix, Arizona 85009.

KONAN™

1448 N. 27th Avenue • Phoenix, Arizona 85009 • 602-269-2649

Circle 205 on inquiry card.

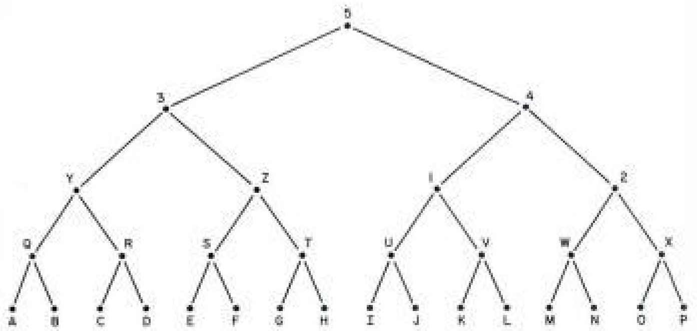


Figure 4: *Simplified version of the game tree that assumes each player has only two possible moves.*

win, no matter what White does, and for exactly the same reason as before.

In most cases, however, the game tree will be far from complete. In chess, for example, you might be in the middle of the game, and neither White nor Black can win the game in the next twenty-five moves. You can still use game trees, but in a slightly different way. The first thing to do is code your knowledge as to when one position is better than another in terms of material gained and lost. For example, if White captures a pawn and loses a Bishop, or captures a Knight and loses a Rook, then Black is obviously ahead. But what if White captures the Queen and loses both Rooks? Is that good or bad? What if White captures two pawns, but loses a Knight?

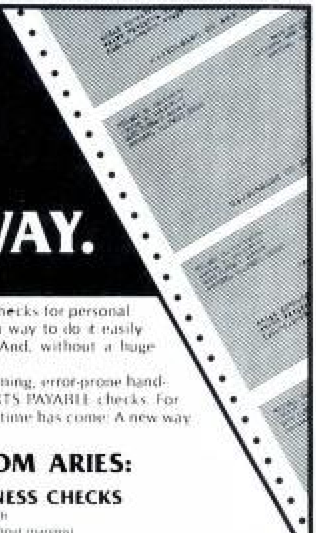
The usual pawn and piece values are: Queen = nine pawns, Rook = five pawns, Bishop and Knight are three pawns apiece. Greatly improved tables of values have been constructed; table 1 is a reprint of values (in abridged form) from R M Hyatt, the author of a chess program called BLITZ. Through the use of such a table, you can derive, for any position, a total numerical score that represents the value of that position. The function which computes this score is called the *evaluation function* corresponding to the given table.

You might think that with such an evaluation function there would be no further need for game trees. You could simply try all of the possible moves, and then choose the one with the largest value of the evaluation function. This, however, would lead to a very bad chess-playing program, rather like someone who had been playing for only a few months. The reason, of course, is that the evaluation function is only an approximation. It is very easy to lose a piece after you have made what seems to be the best move according to your evaluation function, because you have not looked far enough ahead. The best game programs use a combination of game trees and an evaluation function, together with the special technique of *alpha-beta pruning*, the subject of this article.

Once more I will set up an artificially small and simple game tree, in order to illustrate how this works. Consider the game tree of figure 5, which is exactly the same as the game tree of figure 4 except that a value of the evaluation function at each of the leaves of the tree has been specified. The evaluation function at the branches has *not* been specified, because this will be computed in a different way. Specifically, look at the leaves A and B. Since the value of the function is 26 at A, and 37 at B, you can conclude that, since it is Black's turn to play, at the branch Q Black will play to branch A. (This move assumes that the higher the value of the evaluation function, the better the position is for White, and the worse

CHECKS IN MICRO-QUANTITIES?

NOW THERE'S A WAY.



Wondering how to use your micro to print checks for personal **and** business use? Now, from ARIES, there is a way to do it easily with **continuous checks in small quantities.** And, without a huge investment.

Micro check printing eliminates time consuming, error-prone hand-writing of PERSONAL, PAYROLL, or ACCOUNTS PAYABLE checks. For home or small business use, it's an idea whose time has come! A new way to make your micro pay off.

NOW AVAILABLE FROM ARIES:

PERSONAL CHECKS	**BUSINESS CHECKS**
8½" width	8½" width
(7½" without margins)	(7½" without margins)
3½" depth	7" depth Includes 3½" voucher, 3½" check)
200 @ $39.00	200 @ $48.00
500 @ $79.00	500 @ $95.50
	Use for either Payroll or Accounts Payable

Includes imprinting of payer at upper left of check—4 lines of 30 positions each, bank name and address, transit and routing numbers, and the M.I.C.R. encoding at the bottom.

Brochure and Samples Available Upon Request. Prices Good thru 1979

Include BANK M.I.C.R. Encoding Sheet With Order

NO ONE ELSE OFFERS MICRO CHECK-PRINTING SO EASILY. WITH ARIES YOUR MICRO PAYS OFF.

ARIES Computer Products
P.O. Box 7932 B Eugene OR 97401
Price include freight anywhere in Continental U.S.

Circle 14 on inquiry card.

the position is for Black. Black will make the move that gives the *lower* evaluation function value. Again, this is only an approximation, but it becomes a better one as the tree gets larger.)

In the same way you may conclude that, since it is Black's turn to move, at branch R Black will move to branch D, since 28 is less than 29. Let us go back to branch Y. Here it is White's turn to play, and White wants to make the move that results in the *highest* value of the evaluation function. Does this mean 37, the largest of the four values at A, B, C, and D? No, it does not. If White plays to Q, Black will play to A. If White plays to R, Black will play to D. Therefore, you should compare only A and D. Since 28 is larger than 26, White should play from Y to R.

This potential source of confusion suggests that you should mark the nodes Q, R, S, T, and so on, with the *expected* evaluation function values (ie: the values that would ensue if Black makes the best play, in a highly approximate sense, on the next move). In this case Q would receive the value 26, R would receive the value 28, and in general each node would receive the *lowest* of the values of the nodes below it. This, of course, is only because it is Black's turn to move. On the next level up, it is White's turn to play, and you can mark each of the nodes Y, Z, 1, and 2 with the *highest* of the values of the nodes below it, because White now wants to make the ultimate value of the evaluation function as large as possible. Continuing this all the way to the top of the tree, you get the situation illustrated in figure 6. The expected value for White at the top of the tree is 25. By following the figure 25 down through the tree, you will see that, at this point in the game, White is expected to move to node 4, Black to reply by moving to node 1, White to then move to U, and Black to play to J.

This does not, of course, have to be what actually happens in the game. Black might be a poor player, and play to node 2 instead of node 1, or Black might discover, upon looking more moves ahead, that node 2 is actually a better play than node 1. This tends to happen in actual games. As you look further ahead (ie: as you consider trees with greater and greater numbers of levels), expected moves at all levels, even the top level, can change.

At this point a very important question is raised: is it really necessary to generate this whole tree? It would be nice to find certain nodes that do not have to be constructed.

Consider the situation at node Z. White has two possible moves: one to node S and one to node T. At node S, White gets a score of at least twenty-two on the next move. Is this a better move for White than the move to node T? To determine the answer, look at node T. The first thing you will see is that if White moves to node T, then Black can move to node G. If Black does that, White ends up with a score of only thirteen. By this point you already know what White should not move to node T because he can do better by moving to node S.

Now look at node H. If White moves to node T, then Black could also move to node H, leaving White with a score of eleven. This is a better move for Black than the move to node G. The point is that *this does not matter*. As soon as you look at node G, you know that White should not move to node T. When you are aware of this it does not matter what

Capturing the Queen	9000
Capturing a Rook	5000
Capturing a Knight or Bishop	3000
Capturing a pawn	1000
Doubled pawns	−30
Tripled pawns	−100
Isolated pawns	−90
Two pawns next to each other	10
One pawn guarding another	36
Knight on opponent's side of the board	40
Same, with pawn guarding it	60
Bishop on strong diagonal	24
Rook on open file	60
Doubled Rooks on open file	170
Rook behind passed pawn	60
Rook on seventh rank, two unmoved opposing pawns	100
Rook on seventh rank, three unmoved opposing pawns	200
Rook on seventh rank, four unmoved opposing pawns	300
Rook moved before castling has occurred	−200
King moved before castling has occurred	−200
Castled King	300
Piece or pawn moved twice in the opening	−30
Taking two moves instead of one to get to a square	−30
Knight never moved	−36
Knight in front of King's pawn or Queen's pawn	−120
Bishop never moved	−20
Bishop in front of King's pawn or Queen's pawn	−120

Table 1: *An abbreviated table of the approximate numerical values assigned to a variety of possible moves.*

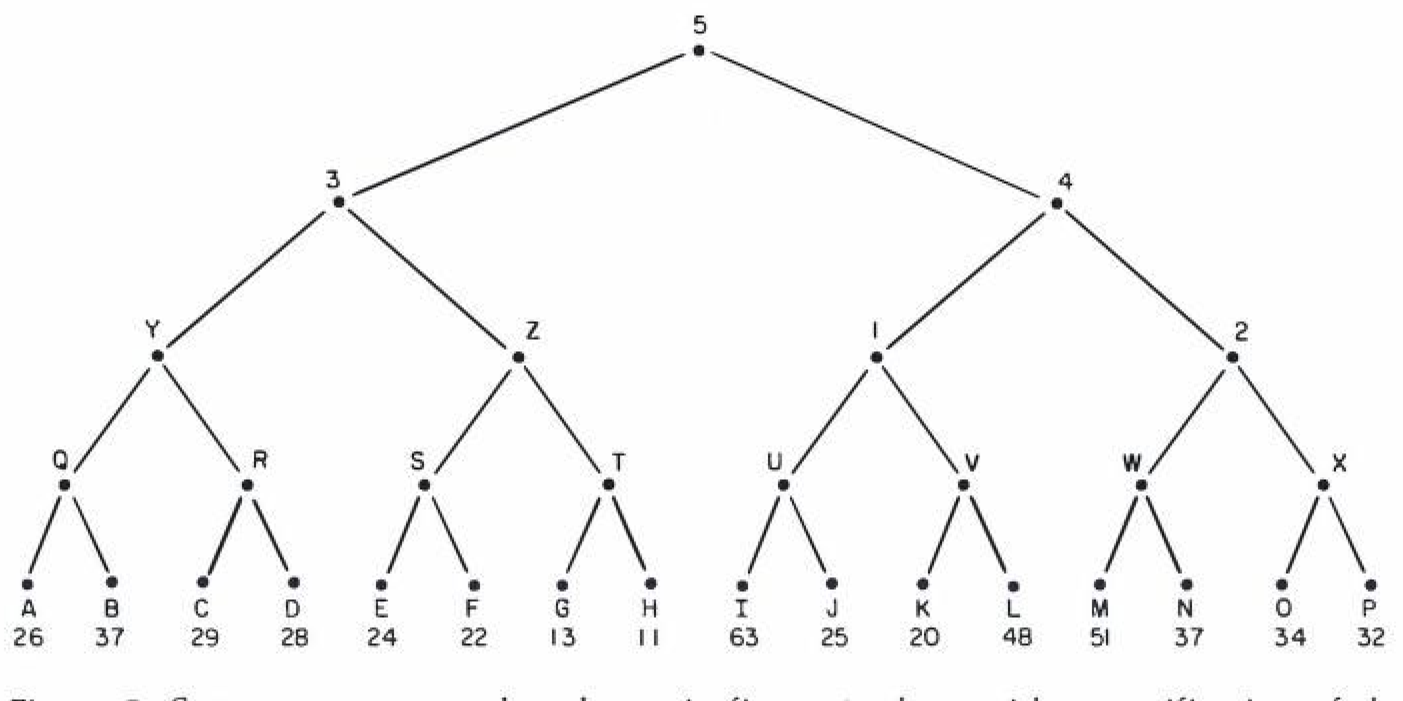


Figure 5: *Same game tree as that shown in figure 4, along with a specification of the evaluation function at each leaf of the tree.*

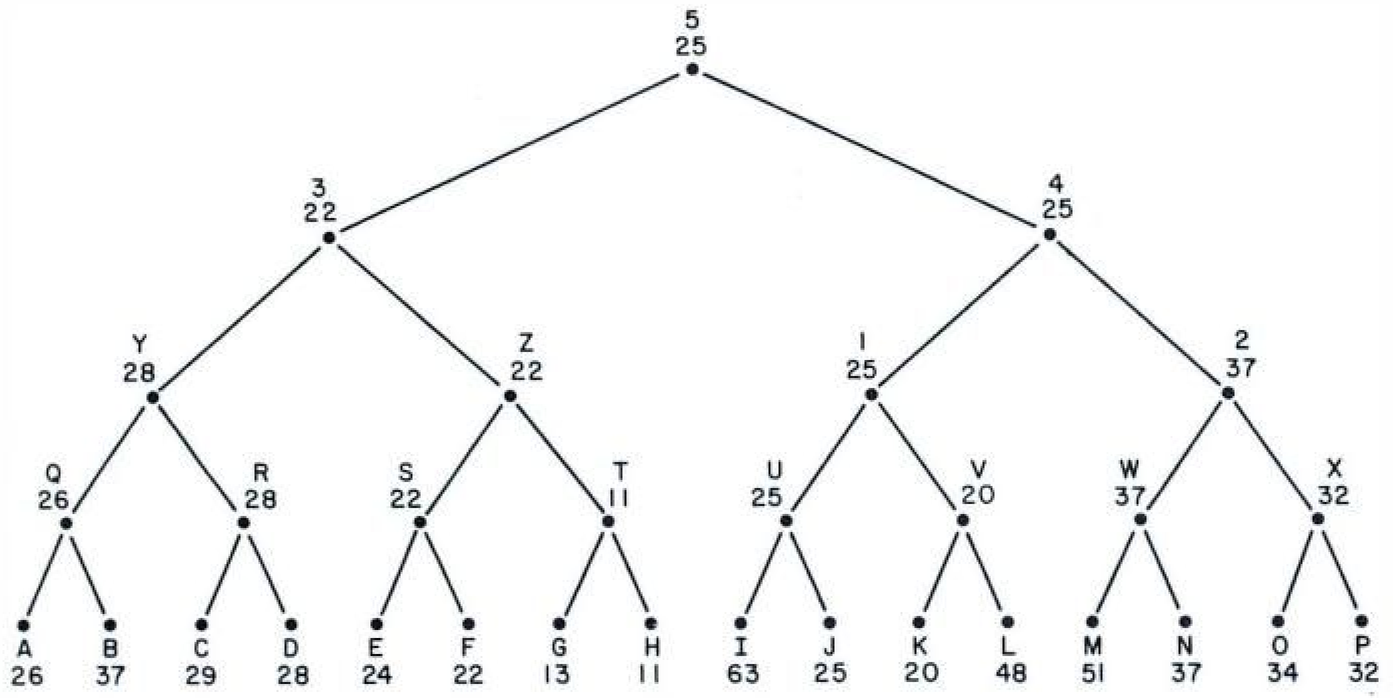


Figure 6: *A more informative version of the game tree shown in figures 4 and 5. Here the expected evaluation function values are shown at each of the nodes.*

score node H has—in fact, you do not have to generate node H at all. This kind of logic can be applied to either

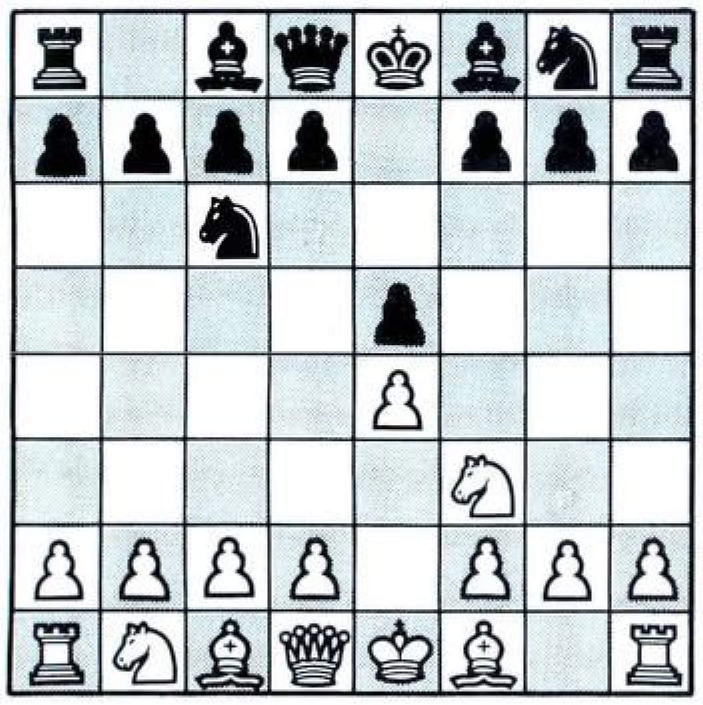


Figure 7: *A simple example to illustrate the principle of alpha-beta pruning. It is now White's turn to move. An obvious bad move would be NxP. Black's reply would be NxN, and White would have captured a pawn but lost a Knight.*

player; it is called *alpha cutoff* in a case like this, where it is White's original move that is being considered (as at node Z here). It is called *beta cutoff* when it is Black's original move that is being considered. *Alpha-beta pruning* is the combination of alpha cutoff and beta cutoff within the general framework described here.

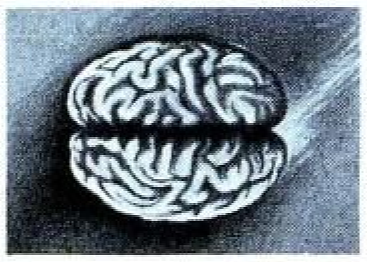
For an example of beta cutoff, look at node 4. It is Black's turn to move. By considering node 1 and all the nodes beneath it (that is, nodes U, V, I, J, K, and L), you will note that Black can eventually expect a score of twenty-five if he moves to node 1. The next question is whether or not a move to node 2 would be any better for Black. Suppose Black moves to node 2, and that White moves to node W. By analyzing the nodes (M and N) beneath node W, you will find that Black can achieve a score of either fifty-one or thirty-seven. Black would naturally choose thirty-seven, that is, node N. But if that is the best that Black can do, then the answer to the original question must be no; that is, a move from node 4 to node 2 would *not* be any better for Black than a move to node 1. Once you know this, it is not necessary to consider node X at all and, more important, you do not have to consider nodes O or P either. In other words, you have pruned not just a single leaf, but a branch with leaves below it.

An informal example of alpha-beta pruning is given in figure 7. Here it is White's turn to move. White has many possible moves, but an obvious *bad* move for White is NxP. In order to determine that this move is bad, it is not necessary to figure out Black's best move; it is only necessary to note that Black can move NxN. Any other possible moves need not be considered as long as White has *any* move that does not result in the loss of a piece, and as long as NxP is not really a viable sacrifice. ■

Glossary

alpha-beta pruning: *In order to guarantee a winning strategy an entire tree search of a complete game tree would be necessary. Alpha-beta pruning is an algorithm devised to optimize the use of game trees by reducing the number of branches needed to be searched.*

game tree: *A graphic representation of the decision making process involved in a sequence of moves between two opponents. A complete game tree is a representation in which all the terminal nodes correspond to the end of the game.*



8086 Boards

CPU with Vectored Interrupts	**$650.**
PROM-I/O	**$495.**
RAM 8K x 16/16K x 8	**$395.**

ANALOG Boards

A/D 16 Channel, **$495.**
12 Bit, High Speed

D/A 4 Channel, **$395.**
12 Bit, High Speed

VIDEO DIGITIZATION

Real Time Video **$850.**
Digitizer and Display

Computer Portrait System **$4950.**

S-100 Boards

Video and/or Analog Data Acquisition Microcomputer Systems

TECMAR INC.

The High Performance S-100 People

TECMAR, INC.
23414 Greenlawn • Cleveland, OH 44122
(216) 382-7599

Circle 364 on inquiry card.