International Journal of Parallel Programming, Vol. 15, No. 2, 1986

Experiments on Alternatives to Minimax

Dana Nau,[1] Paul Purdom, and Chun-Hung Tzeng

Received August 1984; revised August 1986

In the field of Artificial Intelligence, traditional approaches to choosing moves in games involve the use of the minimax algorithm. However, recent research results indicate that minimaxing may not always be the best approach. In this paper we report some measurements on several model games with several different evaluation functions. These measurements show that there are some new algorithms that can makes significantly better use of evaluation function values than the minimax algorithm does.

KEY WORDS: Artificial intelligence; decision analysis; games trees; minimax, search.

"There's something the matter with minimax in the presence of error."—Tom Truscott, co-author of *Duchess*, in his spoken presentation of Ref. 1.

1. INTRODUCTION

This paper is concerned with how to make the best use of evaluation function values to choose moves in games and game trees. The traditional approach used in Artificial Intelligence is to combine the values using the minimax algorithm. Previous work by Nau,[2,14] Pearl,[4] and Tzeng and Purdom[5,6] has shown that this approach is not always best. In this paper we report some measurements on several model games with several different evaluation functions. These measurements show that there are some new algorithms that can make significantly better use of evaluation function values than the minimax algorithm does.

[1] This work was supported in part by a Presidential Young Investigator Award to Dana Nau, including matching funds from IBM Research, General Motors Research Laboratories, and Martin Marietta Laboratories.

0885-7458/86/0400-0163$05.00/0

We consider a game between two players, *Max* and *Min*. The game begins in some starting position. At each position, the player that must move has a finite set of possible moves, each of which leads to a different new position. The players alternate making moves until a *terminal position* is reached where the set of possible moves is empty. We have a *finite* game, so from every position any sequence of moves leads to a terminal position after a finite number of moves. Associated with each terminal position g is a number v_g, the *value* of g. Max's goal is to get to a terminal position with the highest possible value, while Min's goal is to get to a terminal position with he lowest possible value. Each player has perfect information concerning the current position, the possible moves, and the value of each terminal position.

Associated with each position of a game is the *minimax value* of the position. This is the value that will result if each player makes the best possible sequence of moves. The minimax value $V(g)$ for a terminal position g is simply v_g as defined previously. For nonterminal positions, the minimax principle says that if it is Max's move at g, then the minimax value $V(g)$ is given by

$$V_{\text{Max}}(g) = \max_{i \in S(g)} \{V(i)\} \tag{1}$$

where $S(g)$ is the set of possible positions that can be reached from g by making a single move, and $V(i)$ is the minimax value of the position i. If it is Min's move at g, then $V(g)$ is given by

$$V_{\text{Min}}(g) = \min_{i \in S(g)} \{V(i)\} \tag{2}$$

If Max (or Min, respectively) always chooses a move leading to a position of highest (or lowest) possible minimax value, then each side will always choose moves leading to the best position obtainable for that side under the assumption that the other side chooses moves in the same way. No one can argue with the conclusion that this is the best to choose moves when one's opponent is playing perfectly and one has the computational resources required for a complete minimax calculation.

Most games, however, are nontrivial. No one can calculate the best move in reasonable time. The traditional game playing program, therefore, does the following. It searches ahead for several moves, uses a static evaluation function to *estimate* the values of the resulting positions, and then combines the estimates using Eqs. (1) and (2) to obtain estimates of the values of the various moves that it can make. Many successful programs have been built on this plan. There is, however, no reason to believe that Eqs. (1) and (2) are the best ways to combine the estimated

values of positions. Indeed, Nau[2,14] showed that for some reasonable games and evaluation functions, when the minimax Eqs. (1) and (2) are used to combine estimates the quality of the move selected gets worse as the search is made deeper. This behavior is called *minimax pathology.*

Pearl[4] suggested that one should consider product propagation as a way to combine values from an evaluation function. Product propagation is intended to be used with values $V(i)$ that are estimates of the probability of a forced win (minimax value = 1), so that $0 \leqslant V(i) \leqslant 1$ for each i. The values $V(i)$ are treated as if they were independent probabilities, and thus Eqs. (1) and (2) are replaced with

$$V_{\text{Max}}(g) = 1 - \prod_{i \in S(g)} (1 - V(i)) \tag{3}$$

and

$$V_{\text{Min}}(g) = \prod_{i \in S(g)} V(i) \tag{4}$$

Nau[7] did some experiments and found that for at least one class of games and evaluation functions, the average quality of move using product propagation was almost always better than with minimax (i.e., the position moved to was more likely to be a forced win) and that product propagation avoided pathology (i.e., deeper search always increased the average quality of the moves).

More recently, Reibman and Ballard[8] investigated an alternative to minimax in which $V_{\text{Min}}(g)$ was defined to be a weighted average of $\{V(i) | i \in S(g)\}$. They showed that under certain conditions, this approach does significantly better than minimax.

Tzeng[5] has found the best way to use the information from heuristic search functions when the goal is to select a move that results in a position where one has a forced win. Under certain conditions (sibling nodes in a game tree are independent, and evaluation functions given the probabilities of forced wins), product propagation is the best method for choosing such a move. Tzeng's theory does not, however, consider whether one will be able to find the follow-up moves needed to produce the foced win. It does little good to move to a forced win position if one makes a mistake on some later move which results in losing the entire game. So, although trying to move to positions where one has a forced win (but doesn't necessarily know how to force the win) leads to good game playing, it does not necessarily lead to the best possible game playing. A complete theory of game playing should allow for the possibility that both players may make a number of mistakes during the game.

In this paper we report the results of some experimental investigations of several methods of propagating estimates of position values. We consider the traditional *minimax propagation*, *product propagation*, and an intermediate method which we call *average propagation*:

$$V_{\text{Max}}(g) = \frac{1}{2}\left[\max_{i \in S(g)} \{V(i)\} + 1 - \prod_{i \in S(g)} (1 - V(i))\right] \tag{5}$$

and

$$V_{\text{Min}}(g) = \frac{1}{2}\left[\min_{i \in S(g)} \{V(i)\} + \prod_{i \in S(g)} V(i)\right] \tag{6}$$

(*Average propagation* does not return a weighted average of the values of the child nodes as was done in Ref. 8; instead it recursively propagates the average of a minimax and a product.) The reason for interest in methods that are intermediate between minimax propagation and product propagation is as follows.

Minimax propagation is the best way to combine values if one's opinions of the values of previously analyzed positions will not change on later moves. However, real game playing programs reanalyze positions after each move is made, and usually come up with slightly different opinions on the later analyses (because, as the problem gets closer to a position, it is able to search more levels past the position). (Minimax propagation is also known to be the best way to combine values att a node N if those values are the exact values. But if one can obtain exact values, then there is no need for searching at all, and thus no need for combining values.)

Product propagation is the best way to combine values if they are estimates of probabilities of forced wins, if the probabilities of forced wins are all independent, and if no one is going to make any mistakes after the first move. But using estimates (which contain errors) of position values on the first move and then making perfect moves for the rest of the game is equivalent to using an estimator with errors for the first move and a perfect estimator for later moves—which implies a drastic reevaluation of the positions after the first move is made. It is also important to point out that although product propagation propagates the values as if they were independent probabilities, this independence assumption does not hold in most games.

The situation encountered in real game playing is generally somewhere between the two extremes previously described. If a game playing program eventually moves to some node N, then the values computed at each move in the game are progressively more accurate estimates of the value of N.

Although the errors in these estimates decrease after each move, they usually do not drop to zero. Therefore, it should be better to use an approach which is intermediate between the two extremes of minimax propagation and product propagation. There are many possible propagation methods satisfying this requirement, and we chose to study one whose values are easy to calculate.

2. THE GAMES AND THE ALGORITHMS

We now describe three closely related classes of games. In each of these games we assume that the player who makes the last move in the game is Max.

A *P*-game is played between two players. The playing board for the game consists of 2^n squares, numbered from 0 to $2^n - 1$. (We use $n = 10$.) Each square contains a number, either 0 or 1. These numbers are put into the squares before the beginning of the game by assigning the value 1 to each square with some fixed probability p and the value 0 otherwise, independent of the values of the other squares. We use $p = (3 - \sqrt{5})/2 \approx 0.382$, which results in each side having about the same chance of winning (the probability that Min will win from a random position is $(3 - \sqrt{5})/2$ if both sides play perfectly.[3]

To make a move in the game, the first player selects either the lower half of the board (squares 0 to $2^{n-1} - 1$) or the upper half (squares 2^{n-1} to $2^n - 1$). His opponent then selects the lower or upper half of the remaining part of the board. (The rules can be generalized for branching factors greater than 2, but we will be concerned only with the binary case.) Play continues in like manner with each player selecting the lower or upper half of the remaining part of the board until a single square remains. If the remaining square contains a 1 then Max (the player to make the last move) wins; otherwise Min wins.

The game tree for a *P*-game is a complete binary game tree of depth k, with random, identically distributed leaf node values (for example, see Fig. 1). For this reason, the minimax value of a node in a *P*-game is independent of the values of other nodes at the same depth. Such independence does not occur in games such as chess or checkers. In these games, the board positions usually change incrementally, so that each node is likely to have children of similar strength. This incremental variation in node strength is modeled in two different ways in the *N*-games and *G*-games. In *N*-games, it is done by assigning strength values to the nodes of the game tree and determining which terminal nodes are wins and losses on the basis of these strengths. In *G*-games it is done by causing sibling nodes to have most of their children in common (as often occurs in games).

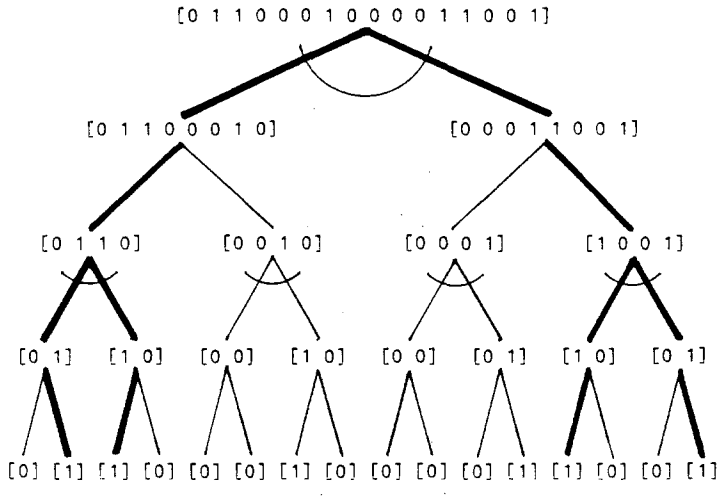

Fig. 1. A game tree for a *P*-game of depth 4. The initial playing board, which appears at the root of the tree, is set up by assigning each square a value of a 1 or 0 at random. Since the depth is even, Max is the second player. Max has a forced win in this particular game, as indicated by the solution tree (drawn in boldface) for Max.

An *N*-game has the same size playing board, the same moves, and the same criterion for winning as a *P*-game, but the initial playing board is set up differently. To set up the board, each arc of the game tree is independently, randomly given the value 1 with probability q or -1 otherwise, for some fixed q (we use $q = 1/2$). The *strength* of a node t in the game tree is defined as the sum of the arc values on the path from t back to the root. A square in the playing board is given the value 1 if the corresponding leaf node of the game tree has positive strength, and the value 0 otherwise (for an example, see Fig. 2).

In contrast to *N*-games and *P*-games, the playing board for a *G*-game is a row of $k+1$ squares, where $k > 0$ is an integer (see Fig. 3). The playing board is set up by randomly assigning each square the value 1 with probability r or the value 0 otherwise, for some fixed r (we use $r = 1/2$). A more (for either player) consists of removing a single square from either end of the row. As with the *P*-games and *N*-games, the game ends when only one square is left. If this square contains a 1, then Max wins; otherwise Min wins.

Note that every node in a *P*-game, *N*-game, or *G*-game is a forced win for one of the two players (Max or Min). This can easily by proved by induction, since *P*-games and *N*-games do not have ties. By a *win* node we

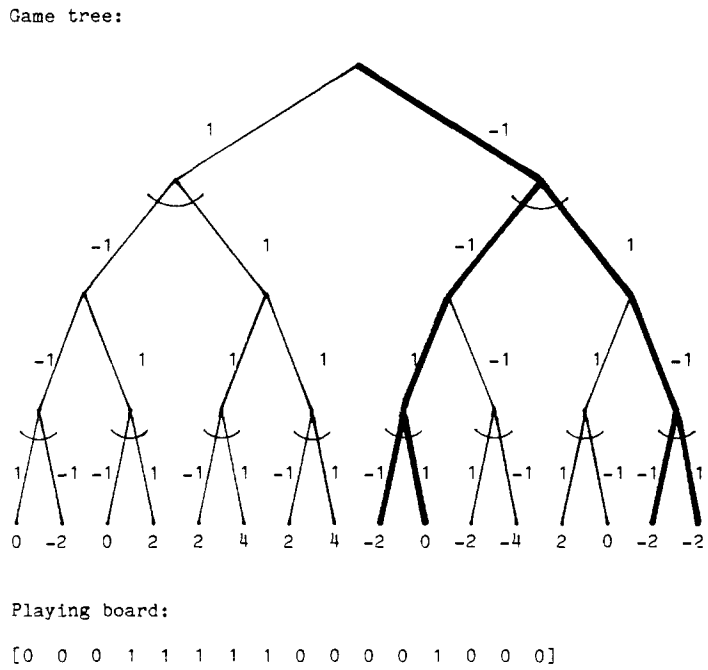

Fig. 2. Setting up the playing board for an N-game of depth 4. A value of 1 or -1 is assigned at random to each arc of the game tree, and the value of each leaf node is taken to be the sum of the arc values on the path back to the root. A square in the playing board is given the value 1 if the corresponding leaf node has a positive value; otherwise it is given the value 0. Since the depth is even, Max is the second player. Min has a forced win in this particular game, as indicated by the solution tree (drawn in boldface) for Min.

mean a node that is a forced win for Max, and by a *loss* node we mean a node that is a forced loss for Max (i.e., a forced win for Min).

Let T be a game tree for a P-game, N-game, or G-game, and t be a node in T. The more "1" squares there are in t the more likely it is that t is a forced win. Thus an obvious evaluation function for T is

$$e_1(t) = \frac{\text{the number of "1" squares in } t}{\text{the number of squares in } t} \tag{7}$$

Investigations in previous papers[3,7] reveal that this is a rather good evaluation function for both P-games and N-games. Not only does it give reasonably good estimates of whether a node is a win or a loss, but it dramatically increases in accuracy as the distance from a node to the end of a game decreases. On the other hand it is not an ideal estimator for use with product propagation, since it does not given the true probability of

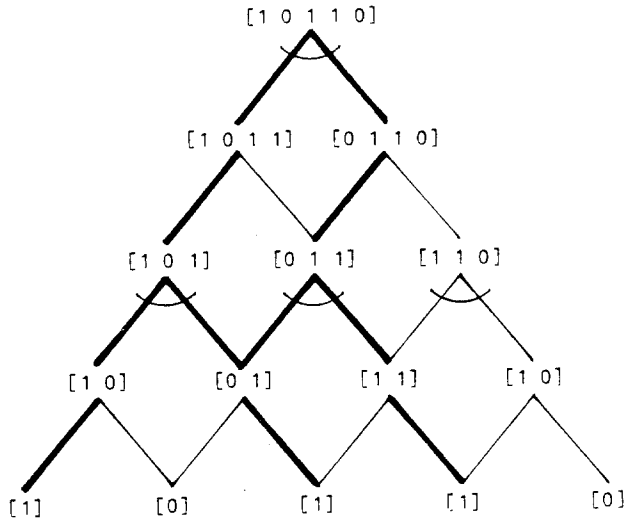

Fig. 3. A game graph for a G-game of depth 4. The initial playing board, which appears at the root of the graph, is set up by assigning each squares a value of 1 or 0 at random. Since the depth is even, Max is the second player. Max has a forced win in this particular game, as indicated by the solution graph (drawn in boldface) for Max.

winning based on the information at hand (the fraction of "1" squares). For example, in P-games it does not vary rapidly enough near $e(t) = (3-\sqrt{5})/2$ (See Fig. 2 in Ref. 9). Instead, this function gives a rough estimate of the probability of winning. This is perhaps typical of the quality of data that real evaluation functions provide.

Three methods of propagating the estimates from evaluation function are compared in this paper: *minimax propagation*, *product propagation*, and a decision rule which is intermediate between these two, which for this paper we call *average propagation.*

We let $M(k, t)$, $P(k, t)$, and $A(k, t)$ be the values propagated by these three rules, where t is a node and k is the depth of node t from the current position. The search strats on depth 0 and proceeds to depth d. The value of the heuristic evaluation function applied to node t is $e(t)$. The three propagation rules are

$$M(k, t) = \begin{cases} e(t) & \text{if } k = d \text{ or } t \text{ is a leaf node} \\ \max_{i \in S(t)}\{M(k+1, i)\} & \text{if } k < d \text{ and it is Max's move} \\ \min_{i \in S(t)}\{M(k+1, t)\} & \text{if } k < d \text{ and it is Min's move} \end{cases} \tag{8}$$

$$P(k, t) = \begin{cases} e(t) & \text{if } k = d \text{ or } t \text{ is a leaf node} \\ 1 - \prod_{i \in S(t)} [1 - P(k+1, i)] & \text{if } k < d \text{ and it is Max's move} \\ \prod_{i \in S(t)} P(k+1, i) & \text{if } k < d \text{ and it is Min's move} \end{cases} \tag{9}$$

$$A(k, t) = \begin{cases} e(t) & \text{if } k = d \text{ or } t \text{ is a leaf node} \\ \frac{1}{2}[\max_{i \in S(t)}\{A(k+1, i)\} + 1 - \prod_{i \in S(t)} [1 - P(k+1, i)]] & \text{if } k < d \text{ and it Max's move} \\ \frac{1}{2}[\min_{i \in S(t)}\{A(k+1, i)\} + \prod_{i \in S(t)} P(k+1, i)] & \text{if } k < d \text{ and it is Min's move} \end{cases} \tag{10}$$

We assume that when t is a terminal node $e(t)$ gives the value of node t.

It is difficult to conclude much about any of these methods by considering how it does on a single game. One cannot tell from a single trial whether a method was good or merely lucky. Therefore we test each method on large sets of P-games, N-games, and G-games. A good propagation method should be able to win more games than any other propagation method.

3. RESULTS AND DATA ANALYSIS

3.1. P-Games Using e_1

Our first set of results is from a set of 1600 randomly generated pairs of P-games. Each pair of games was played on a single game board; one game was played with one player moving first and another was played with his opponent moving first. Of the 1600 game boards, 970 were boards where the first player had a forced win and 630 were boards where the second player had a forced win. The expected results from from our random game generation process were $1600p \approx 611$ forced wins for the second player with a standard deviation of $\sqrt{1600p(1-p)} \approx 18.9$. Our observed deviation from the expected value should occur about 33% of the time. Thus this is a rather typical random sample of games.

For each pair of games we had 10 contests, one for each depth of searching from 1 to 10. Each contest included all 1600 pairs of games. Most game boards were such that the position (first player to move or second player to move) rather than the propagation method determined who won the game, but for some game boards one propagation method was able to win both games of the pair. We call these latter games *critical games.*

For each P-game contest, Table la shows how many pairs were won by a single method (the number of critical games) and how many of those

pairs were won by the first method in the contest. For example, the contest played at search depth 2 between product propagation and minimax propagation contained 472 critical games. Of these, product propagation won 231 games, not quite half.

Table 1b summarizes the raw data from Table 1a. It gives the percentage of the games that the first method won in each P-game contest. A percentage greater than 50% indicates that the first method did better than the second method most of the time. However, if the percentage is neither 0% nor 100%, then for each method we found some games where it did better than its opponent. The results in this table show that for the set of games considered, average propagation was always as good as and often several percent better than either minimax propagation or product propagation. Product propagation was usually better than minimax propagation, but not at all search depths.

An important question is how significant the results are. Even if two methods are equally good on the average, chance fluctuations would usually result in one of the methods winning over half the games in a 1600 game contest. To test the significance of each result, we considered the

Table 1a. Number of Pairs of P-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, with both Players Searching to the Same Depth d using the Evaluation Function e_1[a]

	Product vs. Minimax		Average vs. Minimax		Average vs. Product		
d	Pairs	Wins	Pairs	Wins	Pairs	Wins	Notes
1	0	0	0	0	0	0	[b]
2	472	231	320	181	240	140	
3	569	249	411	218	332	199	
4	597	334	520	331	352	221	
5	577	290	478	308	341	227	
6	567	348	525	385	266	191	
7	424	235	352	229	205	140	
8	324	223	305	236	95	70	
9	0	0	0	0	0	0	[b,c]
10	0	0	0	0	0	0	[b,c]

[a] The results come from Monte Carlo simulations of 1600 game boards each. For each game board and each value of d, a pair of games was played, so that each player had a chance to start first. Out of the 1600 pairs, a pair was counted only if the same player won both games in the pair.

[b] For search depths 1, 9, and 10, both players play identically.

[c] For search depths 9 and 10, both players play perfectly.

Table 1b. Percentage of Pairs of *P*-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, in the Same Games Used for Table 1a[a]

	Product vs. Minimax			Average vs. Minimax			Average vs. Product		
d	%Wins	Significance		%Wins	Significance		%Wins	Significance	
2	48.9%	0.65	?	56.6%	0.019	Avg.	58.3%	0.012	Avg.
3	43.8%	0.0028	Prod.	53.0%	0.23	?	59.9%	0.0003	Avg.
4	55.9%	0.0038	Prod.	63.7%	1×10^{-9}	Avg.	62.8%	2×10^{-6}	Avg.
5	50.3%	0.90	?	64.4%	2×10^{-10}	Avg.	66.6%	9×10^{-10}	Avg.
6	61.4%	6×10^{-8}	Prod.	73.3%	1×10^{-26}	Avg.	71.8%	1×10^{-12}	Avg.
7	55.4%	0.026	Prod.	65.1%	2×10^{-8}	Avg.	68.3%	2×10^{-7}	Avg.
8	68.8	1×10^{-11}	Prod.	77.4%	1×10^{-21}	Avg.	73.7%	4×10^{-6}	Avg.

[a] The significance column gives the probability that the data is consistent with the null hypothesis that each method is equally good. Small numbers (say, 0.05), indicate that the deviation away from 50% in the percentage of wins is unlikely to be from chance fluctuations—and these numbers are followed by the name of the propagation method that did better. Large numbers indicate that from this data one cannot reliably conclude which method is best—and these numbers are followed by '?'s.

Table 1c. The Results Come from a Monte Carlo Simulation Involving 1600 Games for each Value of *k*[a]

depth	height *k*										
d	3	4	5	6	7	8	9	10	11	12	13
1	0.947	0.906	0.842	0.809	0.762	0.729	0.694	0.670	0.643	0.621	0.620
2	1.000	0.935	0.887	0.819	0.779	0.741	0.693	0.672	0.630	0.624	0.626
3	1.000	1.000	0.927	0.853	0.800	0.758	0.701	0.673	0.643	0.622	0.618
4		1.000	1.000	0.897	0.842	0.769	0.721	0.677	0.649	0.620	0.617
5			1.000	1.000	0.885	0.802	0.734	0.693	0.653	0.632	0.628
6				1.000	1.000	0.844	0.776	0.702	0.669	0.626	0.626
7					1.000	1.000	0.824	0.744	0.677	0.643	0.637
8						1.000	1.000	0.790	0.708	0.667	0.633
9							1.000	1.000	0.757	0.687	0.650
10								1.000	1.000	0.733	0.671
11									1.000	1.000	0.724
12										1.000	1.000
13											1.000

[a] The probability that average propagation chooses a "correct" move (the move leading to a forced win at a node having one forced win child and one forced loss child) when searching to depth d using the evaluation function e_1, at a node of heigh k in a *P*-game.

null hypothesis that the number of pairs of wins (among the critical games) was a random event with probability one half. If there were N critical games, then under the null hypothesis, the expected number of wins by the first method would be $N/2$. If the actual number of wins is A, then under the null hypothesis the probability that the number of wins is less than A or more than $N-A$ is

$$2 \sum_{0 \leq i \leq A} \binom{N}{i} p^i (1-p)^{N-i} \tag{11}$$

when $A < N/2$; and it is this expression with A replaced by $N-A$ when $A > N/2$. (For $N > 240$ we approximated Eq. (11) with a normal distribution.) This number is given in the significance column. It gives the probability that a derivation of the observed amount (in either direction) from 50% wins will arise from chance in a contest between equally good methods. Thus when the number in the significance column is high (say, above 0.05), it is quite possible that the observed results arose from chance fluctuations, and the results are not significant. When the number is small, then it is unlikely that the observed result could have arisen from chance fluctuations—and thus one can be rather sure that the method that won over 50% of the games in this sample is actually the better method.

The P-game contest with estimator e_1 show product propagation doing better than minimax propagation at most search depths. Minimax propagation was better for search depth 3. For depths 2 and 5, the results were too close to be sure which method was better. For depths 3, 4, 6, 7, and 8 product propagation clearly did better. It is interesting to notice that on the games tested, minimax propagation did relatively better when the search depth was odd (i.e., the performance for each odd search depth was better than for either of the search depths one more and one less).

These contests also show average propagation to be a clear winner over minimax propagation in P-games when e_1 is used. Only at depth 3 were the results close enough for there to be any doubt. In addition, average propagation was a clear winner over product propagation at all search depths.

Table 1c shows the fraction of the time (at those nodes where it matters which move is chosen) that the average propagation method with estimator e_1 selects a move that leads to a forced win on P-games. A comparison of these figures with the corresponding figures for minimax propagation (Table 3 in Ref. 7) and product propagation (Table 2 in Ref. 7) shows that for most heights and search depths average propagation does the best of these three methods for using estimator e_1 to select nodes that are forced wins.

3.2. *P*-Games Using e_2

Tzeng[6] gives a formula for the probability $p(h, l)$ that a node in a *P*-game is a forced win, given that there are h moves left at node t and that t contains l ones. We have used Tzeng's formula to compute $p(h,l)$ for all $h \leqslant 8$. Since the number of ones in a node t is $2^h e_1(t)$ and the number of zeroes in t is $2^h(1-e_1(t))$, the probability that t is a forced win given the number of ones and zeroes in t is

$$e_2(t) = 2^h p(h, e_1(t)) \tag{12}$$

It is shown in Ref. 5 that for *P*-games product propagation does the best of any equally informed algorithm for selecting nodes that are forced wins *when the evaluation function returns estimates that are the probabilities of forced wins* (estimator e_2). Tables 2a and 2b duplicate the studies done in Tables 1a and 1b, but using the evaluation function e_2 rather than e_1. In these tables, average propagation and product propagation both do better than they did before in comparison to minimax propagation. Average

Table 2a. Number of Pairs of *P*-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, with both Players Searching to the Same Depth *d* using the Evaluation Function e_2[a]

	Product vs. Minimax		Average vs. Minimax		Average vs. Product		
d	Pairs	Wins	Pairs	Wins	Pairs	Wins	Notes
1	0	0	0	0	0	0	[b]
2	376	202	269	154	150	81	
3	424	229	333	199	184	100	
4	525	336	454	315	183	106	
5	474	310	388	257	167	77	
6	530	399	493	381	133	70	
7	338	223	327	214	74	37	
8	307	243	297	239	34	18	
9	0	0	0	0	0	0	[b,c]
10	0	0	0	0	0	0	[b,c]

[a] The results come from Monte Carlo simulations of 1600 game boards each. For each game board and each value of *d*, a pair of games was played, so that each player had a chance to start first. Out of the 1600 pairs, a pair was counted only if the same player won both games in the pair.

[b] For search depths 1, 9, and 10, both players play identically.

[c] For search depths 9 and 10, both players play perfectly.

Table 2b. Percentage of Pairs of *P*-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, in the Same Games Used for Table 2a[a]

	Product vs. Minimax			Average vs. Minimax			Average vs. Product		
d	%Wins	Significance		%Wins	Significance		%Wins	Significance	
2	53.7%	0.15	?	57.2%	0.018	Avg.	54.0%	0.372	?
3	54.0%	0.099	?	60.0	0.014	Avg.	54.3%	0.27	?
4	64.0%	1×10^{-10}	Prod.	69.4%	2×10^{-16}	Avg.	57.9%	0.038	Avg.
5	65.4%	1×10^{-11}	Prod.	66.2%	1×10^{-10}	Avg.	46.1%	0.35	?
6	75.3%	9×10^{-30}	Prod.	77.3%	2×10^{-38}	Avg.	52.6%	0.60	?
7	66.0%	2×10^{-9}	Prod.	65.4%	2×10^{-8}	Avg.	50.0%	1.00	?
8	79.2%	4×10^{-24}	Prod.	80.5%	7×10^{-26}	Avg.	52.9%	0.84	?

[a] The significance column gives the probability that the data is consistent with the null hypothesis that each method is equally good. Small numbers (say, 0.05), indicate that the deviation away from 50% in the percentage of wins is unlikely to be from chance fluctuations—and these numbers are followed by the name of the propagation method that did better. Large numbers indicate that from this data one cannot reliably conclude which method is best—and these numbers are followed by '?'s.

propagation appears to do better than product propagation at most search depths, but the results *are not statistically significant* except at search depth 4, where they are marginally significant. These results show that product propagation becomes relatively better compared to both minimax propagation and average propagation when better estimates are used for the probability that a node is a forced win.

3.3. *N*-Games Using e_1

Table 3a shows the raw data for *N*-games. The results suggest that for this set of games the averages propagation method of propagation may again be the best, but the differences among the methods are much smaller. Table 3b gives the percentage of wins for each method and the significance. This time minimax propagation is better than product propagation for search depths 3 and 4 (and probably 2). Average propagation may be better than minimax propagation at larger search depths (all the results were above 50%(but one can not be sure based on this data. Average propagation is better than product propagation for all search depths except 8, where the results are inconclusive. It is more difficult to draw definite

Table 3a. Number of Pairs of *P*-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, with both Players Searching to the Same Depth *d* using the Evaluation Function e_1[a]

	Product vs. Minimax		Average vs. Minimax		Average vs. Product		
d	Pairs	Wins	Pairs	Wins	Pairs	Wins	Notes
1	0	0	0	0	0	0	[b]
2	128	53	80	41	59	40	
3	109	41	70	33	51	35	
4	107	42	72	39	45	37	
5	57	27	42	25	19	15	
6	66	28	45	27	11	10	
7	20	10	16	11	7	7	
8	12	6	7	5	5	4	
9	0	0	0	0	0	0	[b,c]
10	0	0	0	0	0	0	[b,c]

[a] The results come from Monte Carlo simulations of 1600 game boards each. For each game board and each value of *d*, a pair of games was played, so that each player had a chance to start first. Out of the 1600 pairs, a pair was counted only if the same player won both games in the pair.

[b] For search depths 1, 9, and 10, both players play identically.

[c] For search depths 9 and 10, both players play perfectly.

Table 3b. Percentage of Pairs of *P*-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, on the Same Games Used for Table 3a[a]

	Product vs. Minimax			Average vs. Minimax			Average vs. Product		
d	%Wins	Significance		%Wins	Significance		%Wins	Significance	
2	41.4%	0.063	?	51.2%	0.91	?	67.8%	0.0086	Avg.
3	37.6%	0.012	Mmax.	47.1%	0.72	?	68.6%	0.011	Avg.
4	39.3%	0.033	Mmax.	54.2%	0.56	?	82.2%	2×10^{-5}	Avg.
5	47.4%	0.79	?	59.5%	0.28	?	78.9%	0.019	Avg.
6	50.9%	1.00	?	60.0%	0.23	?	90.9%	0.012	Avg.
7	50.0%	1.00	?	68.8%	0.21	?	100.%	0.016	Avg.
8	50.0%	1.00	?	71.4%	0.45	?	80.0%	0.38	?

[a] The significance column gives the probability that the data is consistent with the null hypothesis that each method is equally good. Small numbers (say, 0.05), indicate that the deviation away from 50% in the percentage of wins is unlikely to be from chance fluctuations—and these numbers are followed by the name of the propagation method that did better. Large numbers indicate that from this data one cannot reliably conclude which method is best—and these numbers are followed by "?"s.

[b] Since these numbers are below 0.05, they are considered significant.

[c] Since these numbers are above 0.05, they are not considered significant.

conclusions for N-games partly because there is such a low percentage of critical games.

No one has yet found the best way to propagate estimates for N-games. As was the case with P-games, the probability that a node is a forced win given a search to some depth d depends on the values of all of the tip nodes of the search tree.[6] But in N-games, the values of the various nodes are not independent, so the calculation is much more difficult than for P-games. Since the product propagation rule treats the values of the nodes as if they were independent probabilities, product propagation is not the best way to use the estimates.

3.4. G-Games Using e_1

In the case of G-games, it was possible to come up with exact values rather than Monte Carlo estimates. This is because there are only $2^{11} = 2048$ distinct initial boards for G-games of depth 10 (as opposed to $2^{2^{10}}$ distinct initial boards for P-games or N-games of depth 10), and thus it was possible to enumerate all possible G-games and try out the three decision methods on all of them. The results of this experiment are given in Tables 4a and 4b. Table 4a gives the exact percentages of games won in

Table 4a. Number of G-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, with Both Players Searching to the Same Depth d Using the Evaluation Function e_1[a]

	Product vs. Minimax			Average vs. Minimax			Average vs. Product		
d	Wins	Prcnt	Better	Wins	Prcnt	Better	Wins	Prcnt	Better
1	2048	50.0%	*	2048	50.0%	*	2048	50.0%	*
2	2405	58.7%	Prod.	2405	58.7%	Avg.	2048	50.0%	—
3	2368	57.8%	Prod.	2368	57.8%	Avg.	2048	50.0%	—
4	2471	60.3%	Prod.	2471	60.3%	Avg.	2048	50.0%	—
5	2363	57.7%	Prod.	2368	57.8%	Avg.	2054	50.1%	Avg.
6	2307	56.3%	Prod.	2306	56.3%	Avg.	2044	49.9%	Prod.
7	2220	54.2%	Prod.	2222	54.2%	Avg.	2046	50.0%	—
8	2094	51.1%	Prod.	2094	51.1%	Avg.	2048	50.0%	—
9	2048	50.0%	b,c	2048	50.0%	b,c	2048	50.0%	b,c
10	2048	50.0%	b,c	2048	50.0%	b,c	2048	50.0%	b,c

[a] For each value of d, all 2048 G-game boards of depth 10 were tried, and each player was given a chance to start first, for a total of 4096 games.
[b] For search depths 1, 9, and 10, both players play identically.
[c] For search depths 9 and 10, both players play perfectly.

Table 4b. Number of Pairs of *G*-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, in the Same Games Used for Table 4a[a]

	Product vs. Minimax			Average vs. Minimax			Average vs. Product			
d	Pairs	Wins	Prcnt	Pairs	Wins	Prcnt	Pairs	Wins	Prcnt	Notes
1	0	0		0	0		0	0		[b]
2	421	389	92.4%	421	389	92.4%	0	0		
3	480	400	83.3%	480	400	83.3%	0	0		
4	545	484	88.8%	545	484	88.8%	0	0		
5	475	395	83.2%	480	400	83.3%	6	6	100%	
6	391	325	83.1%	386	322	83.4%	4	0	0%	
7	288	230	79.9%	290	232	80.0%	2	0	0%	
8	126	86	68.3%	126	86	68.3%	0	0		
9	0	0		0	0		0	0		[b,c]
10	0	0		0	0		0	0		[b,c]

[a] Out of the 2048 pairs of games, a pair was counted in this table only if the same player won both games in the pair.

[b] For search depths 1, 9, and 10, both players play identically.

[c] For search depths 9 and 10, both players play perfectly.

competitions by minimax propagation, product propagation, and average propagation. For comparison with Tables 1a, 2a, and 3a, Table 4b gives the number of pairs of games won. As can be seen, product propagation and average propagation both did somewhat better than minimax propagation on *G*-games, and did about the same as each other.

3.5. *G*-Games Using e_3

For *G*-games it has been shown [see Ref. 10] that whether or not a node g is a forced win depends solely on the values of the two or three squares in the center of g. Thus the evaluation function e_1 is not a very good one for *G*-games, since it does not give much weight to the values of these squares. For this reason, we constructed an evaluation function e_3 which gives considerably more weight to the squares at the center of the board than the ones at the edge of the board. The function e_3, which is considerably more accurate than e_1 on *G*-games, is defined as

$$e_3(t) = \frac{1}{2^n} \sum_{0 \leqslant i \leqslant n} \binom{n}{i} t_i \tag{13}$$

where t_i is the value of the ith square in t.

Table 5a. Number of *G*-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, with Both Players Searching to the Same Depth *d* Using the Evaluation Function e_4[a]

	Product vs. Minimax			Average vs. Minimax			Average vs. Product		
d	Wins	Prcnt	Better	Wins	Prcnt	Better	Wins	Prcnt	Better
1	2048	50.0%	[b]	2048	50.0%	[b]	2048	50.0%	[b]
2	2030	49.6%	Mmax.	2030	49.6%	Mmax.	2048	50.0%	—
3	2043	49.9%	Mmax.	2051	50.1%	Avg.	2048	50.0%	—
4	1952	47.7%	Mmax.	1980	48.3%	Mmax.	2072	50.6%	Avg.
5	1920	46.9%	Mmax.	1979	48.3%	Mmax.	2116	51.7%	Avg.
6	1924	470%	Mmax.	1968	48.0%	Mmax.	2100	51.3%	Avg.
7	1944	47.5%	Mmax.	1992	48.6%	Mmax.	2116	51.7%	Avg.
8	1968	48.0%	Mmax.	2008	49.0%	Mmax.	2088	51.0%	Avg.
9	2048	50.0%	[b,c]	2048	50.0%	[b,c]	2048	50.0%	[b,c]
10	2048	50.0%	[b,c]	2048	50.0%	[b,c]	2048	50.0%	[b,c]

[a] For each value of *d*, all 2048 *G*-game boards of depth 10 were tried, and each player was given a chance to start first, for a total of 4096 games.
[b] For search depths 1, 9, and 10, both players play identically.
[c] For search depths 9 and 10, both players play perfectly.

Table 5b. Number of Pairs of *G*-Games Won by Product Propagation against Minimax Propagation, Average Propagation against Minimax Propagation, and Average Propagation against Product Propagation, in the Same Games Used for Table 5a[a]

	Product vs. Minimax			Average vs. Minimax			Average vs. Product			
d	Pairs	Wins	Prcnt	Pairs	Wins	Prcnt	Pairs	Wins	Prcnt	Notes
1	0	0		0	0		0	0		[b]
2	72	27	37.5%	72	27	37.5%	0	0		
3	37	16	43.2%	45	24	53.3%	0	0		
4	096	0	0.0%	68	0	0.0%	24	24	100%	
5	128	0	0.0%	69	0	0.0%	68	68	100%	
6	124	0	0.0%	80	0	0.0%	52	52	100%	
7	104	0	0.0%	56	0	0.0%	68	68	100%	
8	80	0	0.0%	40	0	0.0%	40	40	100%	
9	0	0		0	0		0	0		[b,c]
10	0	0		0	0		0	0		[b,c]

[a] Out of the 2048 pairs of games, a pair was counted in this table only if the same player won both games in the pair.
[b] For search depths 1, 9, and 10, both players play identically.
[c] For search depths 9 and 10, both players play perfectly.

Tables 5a and 5b duplicate the data given in Tables 4a and 4b, but using e_3 rather than e_1. Although average propagation and product propagation still do about equally well, this time both do somewhat worse than minimax propagation. One explanation for this is the following. Since e_3 gives more weight to the squares in the center of the board, and since these squares are the last likely ones to be removed as the game progresses, the evaluations given by e_3 will change less dramatically as the game progresses than the evaluations given by e_1. But as pointed out in the introduction to this paper, minimax propagation is the best way to combine values if one's opinion of each position will not change as the game progresses. Thus we would expect the observed result that minimax propagation does better in relation to product propagation and average propagation when using e_3 than when using e_1.

4. CONCLUSION

We tested three methods of propagating the estimates generated by heuristic search functions: minimax propagation, product propagation, and average propagation. We tested the methods on three types of games: *P*-games, *N*-games, and *G*-games. For *P*-games and *G*-games we considered two different heuristic search functions. The main conclusions are that the method used to back up estimates has a definite effect on the quality of play, and that the traditional minimax propagation method is often not the best method to use.

On the games we tested, the differences in performance are often small because in many cases each method selects the same move. Often the result of a contest depends on which propagation method is used only for a small fraction of the games. For those critical games where the propagation method matters, one method will often be much better than the other.

There is no one method that is best for propagating estimates. Which method of propagation works best depends on both the estimator and the game. For example, when playing *G*-games with a naive estimator, product propagation and average propagation each play significantly better than minimax propagation (winning 60% of the games and 89% of the critical games at lookahead 4, for example). On the other hand, when a better estimator is used, minimax propagation does better than either product propagation or average propagation (winning 52% of the games and 100% of the critical games at lookahead 4). One cannot conclude, however, that use of a better estimator automatically favors minimax propagation. For *P*-games, it has been proven in Ref. 5 that product propagation is the best method of propagating estimates in order to select a move that leads to a winning position—and when using an estimator that

returned the probability of winning, product propagation did quite well. For example, at lookahead 4, it won 55% of the games and 64% of the critical games against minimax propagation. On the other hand, when product propagation used a less good estimator, the results were mixed.

Average propagation was able to do better than product propagation under many conditions. The most interesting test was the series of *P*-games where the better estimator was used. For this series of contest, product propagation is known to be the optimum algorithm if the goal is to always try to move toward a position where a forced win exists.[5] One might think that this is a prefectly good goal, but there is one catch—just because a node is a forced win does not mean that a program will be able to choose the correct sequence of moves to foce the win.

So, how good was the goal in practice? At the sensitivity of our experiments it was pretty good. Although average propagation won more games than product propagation (the perfect algorithm for the goal of making a single good move) at most lookaheads, the amount was not statistically significant except at lookahead 4, where the amount was marginally significant. We can be more than 96% sure that average propagation is better than product propagation at winning 10-level *P*-games when both sides use lookahead 4.

One difference between "real" games and the games that we used for our tests is that real games usually have more moves. Thus it is possible that various alternatives to minimax propagation might do even better in "real" games than they did on the games used in this paper, because there may be more opportunity for small improvements in approach to lead to differences in who wins the game. Thus when designing game programs, it might be a good idea to consider what method of propagating estimates to use[2] rather than just automatically choosing minimax propagation. (Some qualifications of this statement are described later.) Propagation methods that favor positions with more than one good continuation deserve particular consideration. Careful theoretical and experimental studies of propagation methods are justified, for this study shows that improved methods do exist.[3] Tzeng[6] gives the outline of a new theory that addresses these questions, but his results have not yet been applied to the analysis of any game.

One problem with methods other than minimax propagation is that the value of every node has some effect on the final result. Thus methods

[2] Decision analysis books such as Ref. 11 describe a number of possible decision criteria to consider.

[3] In fact, work currently in progress in Ref. 12 indicates that a modified version of product propagation outperforms minimax propagation in the game of Kalah.

such as the alpha-beta pruning procedure cannot be used to speed up the search without affecting the final value computed. Programs for most games use deep searches, and these programs will not be able to make much use of these new methods unless suitable pruning procedures are found. A method is needed that will always expand the node that is expected to have the largest effect on the value.

The games where the new results may have the most immediate application are probabilistic games such as backgammon, where it is not feasible to do deep searches of the game tree. Since alpha-beta pruning does not save significant amounts of work on shallow searches, it is conceivable that such games can profit immediately from improved methods of backing up values.

REFERENCES

1. T. R. Truscott, Minimum Variance Tree Searching, *Proc. First Intl. Symp. on Policy Analysis and Infor. Systems*, Durham, NC, pp. 203–209 (1979).
2. D. S. Nau, Decision Quality as a Function of Search Depth on Game Trees, *J. of the ACM* **30**:687–708 (October 1983).
3. D. S. Nau, The Last Player Theorem, *Artificial Intelligence* **18**:53–65 (1982).
4. J. Pearl, On the Nature of Pathology in Game Searching, *Artificial Intelligence* **20**:427–453 (1983).
5. H. C. Tzeng and P. W. Purdom, A Theory of Game Trees, *Proc. of the National Conf. on Artificial Intelligence*, Washington, D.C., pp. 416–419 (1983).
6. H. C. Tzeng, Ph.D. thesis, Computer Science Department, Indiana University (1983).
7. D. S. Nau, Pathology on Game Trees Revisited, and an Alternative to Minimaxing, *Artificial Intelligence* **21**:221–244 (1983).
8. A. L. Reibman and B. W. Ballard, Non-Minimax Search Strategies for Use against Fallible Opponents, National Conference on Artificial Intelligence, Washington, D.C., pp. 338–342 (1983).
9. J. Pearl, Asymptotic Properties of Minimax Trees and Game-Searching Procedures, *Artificial Intelligence* **14**:13–138 (1980).
10. D. S. Nau, On Game Graph Structure and its Influence on Pathology, *Intl. J. Computer and Info. Sciences* **12**:367–383 (1983).
11. I. H. LaValle, *Fundamentals of Decision Analysis*, Holt, Rinehart, and Winston, New York (1978).
12. P. C. Chi and D. S. Nau, Predicting the Performance of Minimax and Product in Game Tree Searching, *Second Workshop on Uncertainty in Artificial Intelligence*, Philadelphia (to appear).
13. B. Abramson, A Cure for Pathological Behavior in Games that Use Minimax, *Proc. Workshop on Uncertainty and Probability in Artificial Intelligence*, Los Angeles, pp. 225–231 (1985).
14. D. S. Nau, An Investigation of the Causes of Pathology in Games, *Artificial Intelligence* **19**:257–278 (1982).