COMPARISON OF THE MINIMAX AND PRODUCT BACK–UP RULES IN A VARIETY OF GAMES[1]

Ping–Chung Chi[2]
Computer Science Department
University of Maryland
College Park, MD 20742

Dana S. Nau[3]
Computer Science Department, and
Institute for Advanced Computer Studies
University of Maryland
College Park, MD 20742

ABSTRACT

This paper describes comparisons of the minimax back–up rule and the product back–up rule on a wide variety of games, including P–games, G–games, three–hole kalah, othello, and Ballard's incremental game. In three–hole kalah, the product rule plays better than a minimax search to the same depth. This is a remarkable result, since it is the first widely known game in which product has been found to yield better play than minimax. Furthermore, the relative performance of minimax and product is related to a parameter called the rate of heuristic flaw (rhf). Thus, rhf has potential use in predicting when to use a back–up rule other than minimax.

[1] A condensed version of this paper, entitled "Comparing Minimax and Product in a Variety of Games", appears in *Proc. AAAI-87.*

[2] This work has been supported in part by a Systems Research Center fellowship.

[3] This work has been supported in part by the following sources: an NSF Presidential Young Investigator Award to Dana Nau, NSF NSFD CDR–85–00108 to the University of Maryland Systems Research Center, IBM Research, and General Motors Research Laboratories.

1. INTRODUCTION

The discovery of pathological games [Nau, 1980] has sparked interest in the possibility that various alternatives to the minimax back–up rule might be better than minimax. For example, the product rule (originally suggested by Pearl [1981, 1984]), was shown by Nau, Purdom, and Tzeng [1985] to do better than minimax in a class of board splitting games.

Slagle and Dixon [1970] found that a back–up procedure called "M & N" performed significantly better than minimax. However, the M & N rule closely resembles minimax. Until recently, poor performance of minimax relative to back–up rules significantly different from minimax has not been observed in commonly known games such as kalah.

This paper presents the following results:

(1) For a wide variety of games, a parameter called the rate of heuristic flaw appears to be a good predictor of how well minimax performs against the product rule. These games include three–hole kalah, othello, P–games, G–games, and possibly others. This suggests that rhf may serve not only as a guideline for whether it will be worthwhile to consider alternatives to minimax, but also as a way to relate other characteristics of game trees to the performance of minimax and other back–up rules.

(2) In studies of three–hole kalah, the product rule played better than a minimax search to the same search depth. This is the first widely known game in which product has been found to play better than minimax. The product rule still has a major drawback: no tree–pruning algorithm has been developed for it, and no correct pruning algorithm for it can conceivably do as much pruning as the various pruning algorithms that exist for minimax. However, the performance of the product rule in kalah suggests the possibility of exploiting non–minimax back–up rules to achieve better performance in other games.

2. DEFINITIONS

By a **game**, we mean a two person, zero sum, perfect information game having a finite game tree. All of the games studied in this paper satisfy this restriction.

Let G be a game between two players called **max** and **min.** To keep the discussion simple, we assume that G has no ties, but this restriction could easily be removed. If n is a board position in G, let u(.) be the utility function defined as

$$u(n) = \begin{cases} 1 & \text{if n is a forced win node} \\ 0 & \text{if n is a forced loss node.} \end{cases}$$

We consider an evaluation function to be a function from the set of all possible positions in G into the closed interval [0,1]. If e is an evaluation function and n is a node of G, then the higher the value e(n), the better n looks according to e. We assume that every evaluation function produces perfect results on terminal game positions (i.e., e(n) = u(n) for terminal nodes).

If m is a node of G, then the depth d minimax and product values of m are

$$\mathcal{M}(m,d) = \begin{cases} e(m) & \text{if depth(m)=d or m is terminal} \\ \min_n \mathcal{M}(n) & \text{if min has the move at m} \\ \max_n \mathcal{M}(n) & \text{if max has the move at m} \end{cases}$$

$$\mathcal{P}(m,d) = \begin{cases} e(m) & \text{if depth(m)=d or m is terminal} \\ \Pi_n\ \mathcal{P}(n) & \text{if min has the move at m} \\ 1- \Pi_n\ (1-(n)) & \text{if max has the move} \end{cases}$$

where n is taken over the set of children of m.

Let m and n be any two nodes chosen at random from a uniform distribution over the nodes at depth d of G. Let $\uparrow_e(m,n)$ (and $\downarrow_e(m,n)$) be whichever of m and n looks better (or worse, respectively) according to e. Thus if $e(m) > e(n)$, then $\uparrow_e(m,n) = m$ and $\downarrow_e(m,n) = n$. If $e(m) = e(n)$, we still assign values to $\uparrow_e(m,n)$ and $\downarrow_e(m,n)$, but the assignment is at random, with the following two possibilities each having probability 0.5:

(1) $\uparrow_e(m,n) = m$ and $\downarrow_e(m,n) = n$;

(2) $\uparrow_e(m,n) = n$ and $\downarrow_e(m,n) = m$.

Since e may make errors, exhaustive search of the game tree may reveal that $\uparrow_e(m,n)$ is worse than $\downarrow_e(m,n)$, i.e., that

$$u(\uparrow_e(m,n)) < u(\downarrow_e(m,n)).$$

In this case, a **heuristic flaw** has occurred: the evaluation function has failed to give a correct opinion about m and n. The rate of heuristic flaw at depth d, denoted by rhf(d), is defined to be the quantity

$$\Pr[u(\uparrow_e(m,n)) < u(\downarrow_e(m,n))].$$

3. THEORETICAL CONSIDERATIONS

3.1. First Hypothesis

Consider a minimax search terminating at depth d of a game tree. If rhf(d) is small, it is intuitively apparent that this search should perform quite well. The question is whether it will perform better than some other back–up rule.

For simplicity, assume that the game tree is binary. Assume further that it is max's move at some node c, and let m and n be the children of c. Let d be the depth of m and n. Then

$$\begin{aligned}(1)\quad \Pr[u(c)=1] &= \Pr[u(\uparrow_e(m,n))=1 \text{ or } u(\downarrow_e(m,n))=1] \\ &= \Pr[u(\uparrow_e(m,n))=1] + \Pr[u(\downarrow_e(m,n))>u(\uparrow_e(m,n))] \\ &\approx \Pr[u(\uparrow_e(m,n))=1] + \text{rhf}(d).\end{aligned}$$

The smallest possible value for rhf(d) is zero. If rhf(d) is close to zero, then from (1) we have

$$\Pr[u(c)=1] \approx \Pr[u(\uparrow_e(m,n))=1],$$

which says that the utility value of c is closely approximated by the utility value of its best child. But according to the minimax rule, the minimax value of c is the minimax value of the best child. This suggests that in this case one might prefer the minimax back–up rule to other back–up rules.

Consider the case when rhf is large. In general, rhf can take on any value between 0 and 1. But if e is a reasonable evaluation function, and if $\uparrow_e(m,n)$ is a forced loss, this should make it more likely that $\downarrow_e(m,n)$ is also a forced loss. Thus, we assume that

$$\Pr[u(\downarrow_e(m,n))=1 \mid u(\uparrow_e(m,n))=0] \leq \Pr[u(\downarrow_e(m,n))=1].$$

Thus since u(.) must be either 0 or 1,

$$\begin{aligned}\text{rhf} &= \Pr[u(\downarrow_e(m,n))=1 \ \& \ u(\uparrow_e(m,n))=0] \\ &\leq \Pr[u(\uparrow_e(m,n))=0]\,\Pr[u(\downarrow_e(m,n))=1].\end{aligned}$$

Suppose rhf is large, i.e.,

$$\text{rhf} \approx \Pr[u(\uparrow_e(m,n))=0]\, \Pr[u(\downarrow_e(m,n))=1].$$

Then from (1),

$$\Pr[u(c)=1] \approx \Pr[u(\uparrow_e(m,n))=1] + \Pr[u(\uparrow_e(m,n))=0]\, \Pr[u(\downarrow_e(m,n))=1].$$

Thus, if $e(\uparrow_e(m,n))$ and $e(\downarrow_e(m,n))$ are good approximations of $\Pr[u(\uparrow_e(m,n))=1]$ and $\Pr[u(\downarrow_e(m,n))=1]$, then

$$\Pr[u(c)=1] \approx e(\uparrow_e(m,n)) + (1 - e(\uparrow_e(m,n)))\, e(\downarrow_e(m,n)) = 1 - (1 - e(\uparrow_e(m,n)))\,(1 - e(\downarrow_e(m,n))),$$

which is precisely the formula for the product rule given in Section 2. This suggests that when rhf is large, the product rule might give a better backup value for c.

From the above considerations, we may form the following hypothesis.

Hypothesis 1. Suppose we are given a game and several different evaluation functions for that game. When an evaluation function is used that has a small rhf value, minimax should perform better against product than it does when an evaluation function is used that has a large rhf value.

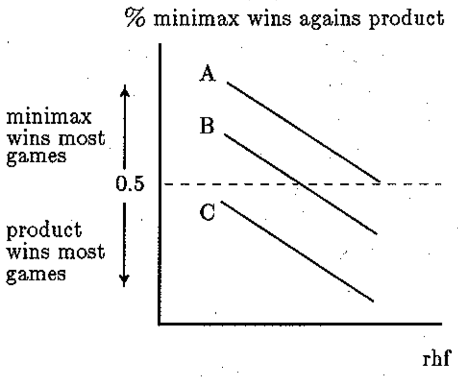

FIGURE 1: Possible curves for Hypotheses 1 & 2

3.2. More Considerations

Hypothesis 1 suggests that, in general, the percentage of wins obtained by minimax against the product rule should decrease monotonically as rhf increases. Therefore, if we plot this percentage against rhf, we should get a curve similar to one of the curves (A, B, C) drawn in Figure 1. For curve A, minimax is always better than product. For Curve C, product is always better than minimax. For Curve B, minimax is better when rhf is small and product is better when rhf is large. Which one of these will more likely be the case?

To answer this question, consider the extreme case where rhf(d)=0. In this case, whenever m and n are two nodes at depth d of G,

$$\Pr[u(\uparrow_e(m,n)) < u(\downarrow_e(m,n))] = 0.$$

Therefore, since there are only a finite number of nodes at depth d, there is a value $k \in (0,1)$ such that for every node m at depth d,

$$u(m) = 1 \text{ if and only if } e(m) \geq k.$$

By mathematical induction, it follows that forced win nodes will always receive minimax values larger than forced loss nodes, so a player using a minimax search will play perfectly.

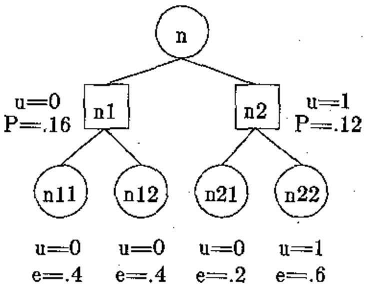

FIGURE 2: A case where product makes the wrong choice.

But if the product rule is used rather than the minimax rule, then the search will not always result in perfect play. For example, consider the tree shown in Figure 2. By looking at the four leaf nodes, it is evident that rhf=0 when k=0.5. Thus, a minimax search at node n would result in a correct decision. However, a product rule search would result in incorrectly choosing the forced loss node n1. This suggests that when rhf is small, the minimax rule should perform better than the product rule.

On the other hand, consider an evaluation function e which returns correct values on terminal nodes, but on nonterminal nodes returns values that are completely unindicative of the true value of a node. This would happen if e always returned the same value (say, 0.5), or if e returned independent identically distributed random values. If e is used in games where the branching factor is not constant, the product rule will tend to choose nodes where one has a wider choice of moves than one's opponent. In this case, it is plausible that the product rule will do slightly better than the minimax rule.

The above arguments are by no means conclusive, but they suggest the following hypothesis:

Hypothesis 2. The minimax rule performs better than the product rule when rhf is small, and worse than the product rule when rhf is large.

According to Hypothesis 2, Curve B in Figure 1 is the correct one, rather than Curves A and C.

4. EMPIRICAL CONSIDERATIONS

To test Hypotheses 1 and 2, we have examined five different classes of games. This section describes the results which have been obtained for these games. For descriptions of the rules of these games, see the Appendix.

4.1. G–Games

A G–game is a board–splitting game investigated in [Nau, 1983], where two evaluation functions e_1 and e_2 were used to compare the performance of minimax and product. The product rule did better than minimax when e_1 was used, and product did worse than minimax when e_2 was used.

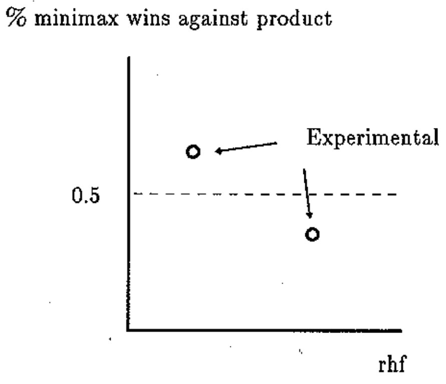

FIGURE 3: Some data points for G–games

It can be proven that for every depth d, rhf(d) is higher using e_1 than it is using e_2. The two data points for e_1 and e_2 can thus be plotted as shown in Figure 3. Thus, these results support both Hypotheses 1 and 2.

4.2. Ballard's Experiments

Ballard [1983] used a class of incremental games with uniform branching factor to study the behavior of minimax and non–minimax back–up rules. One of the non–minimax back–up rules was a weighted combination of the computational schemes used in the minimax and product rules. Among other results, he claimed that "lowering the accuracy of either max's or min's static evaluations, or both, serves to increase the amount of improvement produced by a non–minimax strategy." Thus, product did better against minimax when the less accurate evaluation functions were used.

Since Ballard's paper does not include definitions of the evaluation functions he used, we cannot determine their rhf values. However, rhf is basically a measure of evaluation function accuracy—so we can feel reasonably certain that the less accurate evaluation functions had higher rhf values. This suggests that Ballard's results for minimax versus the product rule can be plotted as the data points labeled "Experimental" in Figure 4. Furthermore, as pointed out in Section 3, it can be proven that on the average, minimax performs better than product when rhf $= 0$. This gives us the data point labeled "Math" in Figure 4. These data points support both Hypotheses 1 and 2.

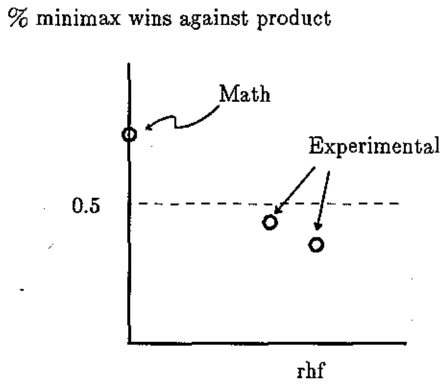

FIGURE 4: Some data points from Ballard's experiments

4.3 Othello

Teague [1985] did experiments on the game of othello, using both a "weak evaluation" and a "strong evaluation." The weak evaluation was simply a piece count, while the strong one incorporated more knowledge about the nature of the game. According to Teague's study, minimax performed better than product 82.8% of the time with the strong evaluation, but only 63.1% of the time with the weak evaluation.

It would be difficult to measure the rhf values for othello, because of the immense computational overhead of determining whether or not playing positions in othello are forced wins. However, since rhf is a measure of the probability that an evaluation function assigns forced win nodes higher values than forced loss nodes, it seems clear that the stronger an evaluation function is, the lower its rhf value should be. Thus, we can feel reasonably certain that Teague's results for minimax versus the product rule can be plotted as the data points labeled "Experimental" in Figure 5. This suggests support for Hypothesis 1. However, it says nothing about Hypothesis 2, because we do not know whether or not the product rule might out–perform minimax for evaluation functions with sufficiently large rhf values.

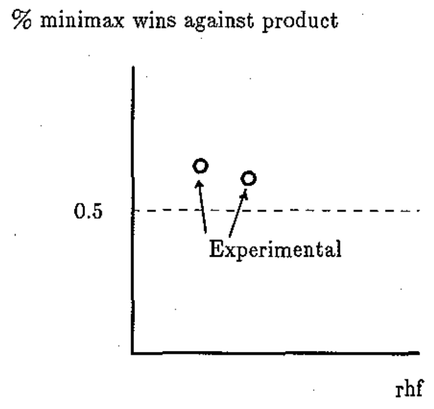

FIGURE 5: Some data points for othello

4.4. P–Games

TABLE 1: Simulation results for P–games of depth 11.

Search depth	% wins for minimax using e1	% wins for minimax using e2
2	51.0%	52.1%
3	52.5%	51.8%
4	49.9%	50.3%
5	50.7%	49.3%
6	46.2%	48.1%
7	46.7%	48.4%
8	44.9%	48.6%
9	47.2%	50.0%

A P–game is a board–splitting game whose game tree is a complete binary tree with random independent assignments of "win" and "loss" to the terminal nodes. P–games have been shown to be pathological when using a rather obvious evaluation function e1 for the games [Nau, 1982]—and in this case, the minimax rule performs more poorly than the product rule [Nau, Purdom, and Tzeng, 1985]. However, pathology in P–games disappears when a stronger evaluation function, e2, is used [Abramson, 1985].

It can be proven that e2 has a lower rhf than e1. Both e1 and e2 return values between 0 and 1, and the only difference between e1 and e2 is that e2 can detect certain kinds of forced wins and forced losses (in which case it returns 1 or 0, respectively).

Let m and n be any two nodes. If $e2(\uparrow_{e2}(m,n)) = 0$, then it must also be that $e2(\downarrow_{e2}(m,n)) = 0$. But it can be shown that $e2(x) = 0$ only if x is a forced loss. Thus $u(\downarrow_{e2}(n, m))=0$, so there is no heuristic flaw. It can also be shown that $e2(x) = 1$ only if x is a forced win. Thus if $e2(\uparrow_{e2}(m,n)) = 1$, then $u(\uparrow_{e2}(m,n))=1$, so there is no heuristic flaw.

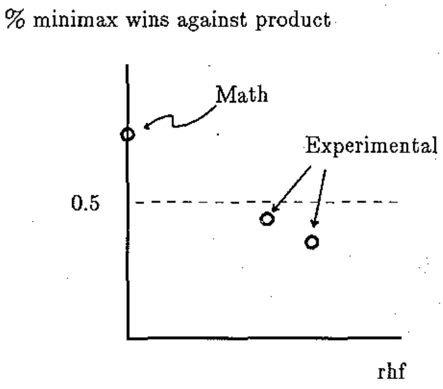

FIGURE 6: Some data points for P–games

Analogous arguments hold for the cases where $e2(\downarrow_{e2}(m,n)) = 0$ or $e2(\downarrow_{e2}(m,n)) = 1$.

The cases described above are the only possible cases where e2 returns a different value from e1. No heuristic flaw occurs for e2 in any of these cases, but heuristic flaws do occur for e1 in many of these cases. Thus, the rhf for e2 is less than the rhf for e1.

We tested the performance of minimax against the product rule using e1 and e2, in binary P–games of depths 9, 10, and 11, at all possible search depths. For each combination of game depth and search depth, we examined 3200 pairs of games. The study showed that for most (but not all) search depths, minimax achieved better performance when using the evaluation function that had the smaller rhf value. For example, Table 1 shows that this is the case for P–games of depth 11, for all search depths except 3 and 5. This supports

Hypothesis 1.

These results also support Hypothesis 2. For example, in P–games of depth 11 with search depth larger than 5, we get the graph sketched in Figure 6.

4.5. Kalah

Slagle and Dixon [1969] states that "Kalah is a moderately complex game, perhaps on a par with checkers." But if a smaller–than–normal kalah playing board is used, the game tree is small enough that one can search all the way to the end of the game tree. This allows one to determine whether a node is a forced win or forced loss. Thus, rhf can be estimated by measuring the number of heuristic flaws that occur in a random sample of games. By playing minimax against product in this same sample of games, information can be gathered about the performance of minimax against product as a function of rhf. To get a smaller–than–normal playing board, we used three–hole kalah (i.e., a playing board with three bottom holes instead of the usual six), with each hole containing at most six stones.

One obvious evaluation function for kalah is the "kalah advantage" used by Slagle and Dixon [1969]. We let e_a be the evaluation function which uses a linear scaling to map the kalah advantage into the interval [0,1].[4] If $P(m,2)$ is computed using $e_a(m)$, the resulting value is generally more accurate than $e_a(m)$. Thus, weighted averages of $e_a(m)$ and $P(m,2)$ can be used to get evaluation functions with different rhf values:

$$e_a^w(m) = w\, e_a(m) + (1-w)\, P(m,2),$$

for w between 0 and 1.

We measured rhf(4), and played minimax against product with a search depth of 2, using the following values for w: 0, 0.5, 0.95, and 1. This was done using 1000 randomly generated initial game boards for three–hole kalah. For each game board and each value of w, two games were played, giving each player a chance to start first. The results are summarized in Table 2.

[4] A preliminary study of rhf [Chi & Nau, 1986] compared minimax to the product rule using e_a in three different variants of kalah. This study, which used a somewhat different definition of rhf than the one used here, motivated the more extensive studies reported in the current paper.

Note that the lowest rhf was obtained with w = 0.5. This suggests that a judicious combination of direct evaluation with tree search might do better than either individually. This idea needs to be investigated more fully.

TABLE 2: Simulation results for kalah

w	rhf(4)	% games won by product	% games won by minimax
1	0.135	63.4%	36.6%
0.95	0.1115	55.5%	44.5%
0	0.08	53.6%	46.4%
0.5	0.0765	51.2%	48.8%

Note also that product performs better than minimax with all four evaluation functions.[5] This suggests that the product rule might be of practical value in kalah and other games. Also, the performance of product against minimax increases as rhf increases. As shown in Figure 7, this provides support for both Hypotheses 1 and 2.

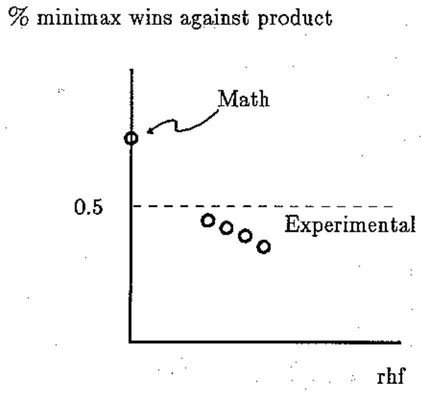

FIGURE 7: Some data points for kalah

[5] Table 2 shows results only for search depth 2. We examined depth 2 to 7 and product rule played better than minimax in all of them except with less statistical significance for depth 3 and 6.

5. P–GAMES WITH VARYING RHF

Section 4 shows that in a variety of games, minimax performs better against product when rhf is low than when rhf is high. However, since the number of data points gathered for each game is rather small, the relationship between rhf and the performance of minimax versus product is still not entirely clear. To investigate further the relationship between rhf and performance of minimax versus product, we did a detailed Monte Carlo study of the performance of minimax against product on binary P–games, using an evaluation function whose rhf could be varied easily.

For each node n, let r(n) be a random value, uniformly distributed over the interval [0,1]. The evaluation function e^W is a weighted average of u and r:

$$e^W(n) = w\ u(n) + (1-w)\ r(n).$$

When the weight $w = 0$, e^W is a completely random evaluation. When $w = 1$, e^W provides perfect evaluations. For $0 \leq w \leq 0.5$, the relationship between w and ee^W is approximately linear (as shown in Figure 8). For $w \geq 0.5$, rhf $= 0$ (i.e., e^W gives perfect performance with the minimax back–up rule).

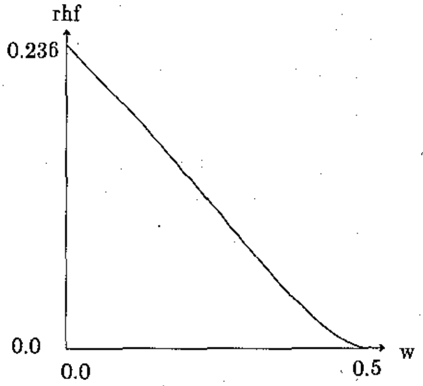

FIGURE 8: The relationship between w and rhf for e^W.

In the Monte Carlo study, 8000 randomly generated initial game boards were used, and w was varied between 0 and 0.5 in steps of

0.01. For each initial board and each value of w, two games were played: one with minimax starting first, and one with product starting first. Both players were searching to depth 2. Figure 9 graphs the fraction of games won by minimax against product, as a function of rhf. Notice that minimax does significantly better than product when rhf is small, and product does significantly better than minimax when rhf is large.[6] Thus, in a general sense, Figure 9 supports our hypotheses about rhf.

But Figure 9 also demonstrates that the relationship between rhf and the performance of minimax against product is not always monotone, and may be rather complex. There are several reasons for this; two of them are described below.

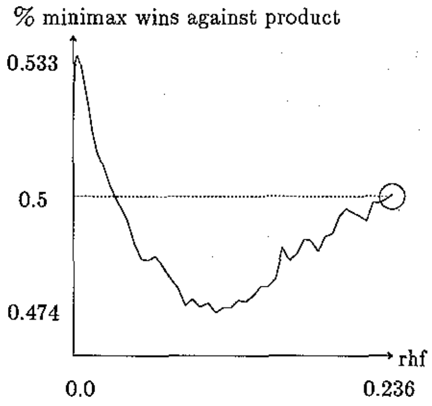

FIGURE 9: Performance of minimax against product using e^{W} as rhf varies.

First, the definition of rhf uses the same notion of a "correct move" as the minimax rule does—and thus we would expect it to be a good predictor of the performance of minimax. However, the relationship between rhf and the performance of the product rule is not as clear. For further studies, it would be advisable to try to formulate a parameter that predicted more closely the performance of the product rule, and use both it and rhf in predicting the performance of minimax versus product.

Second, the argument (in Section 3.2) that the product rule should do better than the minimax rule for extremely high rhf values

[6] Furthermore, the poor performance of minimax when rhf is large corroborates previous studies which showed that product did better than minimax in P-games using a different evaluation function [Nau, Purdom, and Tzeng, 1985].

applies only to games of variable branching factor. Since P–games have a constant branching factor, we can prove mathematically that when the evaluation function returns random values, both the product rule and the minimax rule play randomly, resulting in the circled data point in Figure 9.

6. CONCLUSIONS AND SPECULATIONS

The results presented in this paper are summarized below:

(1) Theoretical considerations suggest that for evaluation functions with low rhf values, minimax should perform better against product than it does when rhf is high. Our investigations on a variety of games confirm this conjecture.

(2) In the game of kalah with three bottom holes, the product rule plays better than a minimax search to the same search depth. This is the first widely known game in which product has been found to yield better play than minimax.

Previous investigations have proposed two hypotheses for why minimax might perform better in some games than in others: dependence/independence of siblings [Nau, 1982] and detection/non–detection of traps [Pearl, 1984]. Since sibling dependence generally makes rhf lower and early trap detection always makes rhf lower, these two hypotheses are more closely related than has previously been realized.

One could argue that for most real games it may be computationally intractable to measure rhf, since one would have to search the entire game tree. But since rhf is closely related to the strength of an evaluation function, one can generally make intuitive comparisons of rhf for various evaluation functions without searching the entire game tree. This becomes evident upon examination of the various evaluation functions discussed earlier in this paper.

There are several problems with the definition and use of rhf. Since it is a single number, rhf is not necessarily an adequate representation for the behavior we are trying to study. Furthermore, since the definition of rhf is tailored to the properties of minimax, it is not necessarily the best predictor of the performance of the product rule. Thus, the relationship between rhf and the performance of minimax versus product can be rather complex (as was shown in Section 5). Further study might lead to better ways of predicting the performance of minimax, product, and other back–up rules.

APPENDIX

A. P–Games and G–Games (adapted from [Nau 1983])

A P–game is played between two players. The playing board for the game is a list of $2^{\mathcal{N}}$ elements (we use $\mathcal{N}=10$). Each element is either -1 or 1. The value of each element is determined before the beginning of the game by making it a 1 with some fixed probability p and a -1 otherwise, independent of the values of the other elements. In order to give each side a reasonable chance of winning, we use

$$p = \frac{(3-\sqrt{5})}{2} \approx 0.382.$$

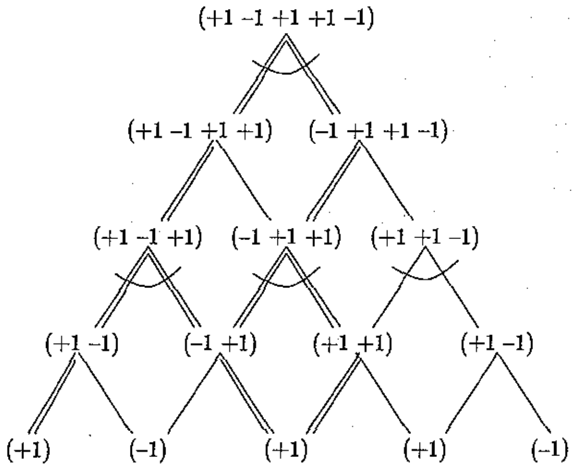

FIGURE 10: A game graph for a G–game of depth 4. The initial board appears at the root. Max, as the second player, has a forced win in this particular game graph, as indicated by the solution graph drawn in double lines.

To make a move in the game, the first player removes either the left half of the list (the first $2^{\mathcal{N}-1}$ elements) or the right half (the last $2^{\mathcal{N}-1}$ elements). His opponent then removes the left or right half of the remaining part of the list. (The rules can be generalized for branching factors greater than 2, but we are concerned only with the binary case.) Play continues in this manner with each player selecting the left or right half of the remaining part of the list until a single element remains. If this element is a 1, then the player who made the last move wins; otherwise his opponent wins.

The game tree for a P–game is a full binary game tree of depth k. Thus the same player always has the last move no matter what course the game takes.

In games such as chess and checkers the game graph is not a tree, since several different nodes may have some of the same children. The G–games described below also have this property.

The playing board for a G–game is a list of k+1 elements, where $k>0$ is an integer. The playing board is set up by randomly assigning each element the value 1 with probability r or the value -1 otherwise, for some fixed r (we use $r=1/2$). A move (for either player) consists of removing a single element from either end of the list (see Fig. 10). As with the P–games, the game ends when only one element is left. If it is a 1, then Max (the player who moved last) wins; otherwise Min wins.

B. Ballard's Experiments

In order to compare different search strategies, Ballard generated random game trees of depth 8, branching factor 4, and search depth 2, using a method given by Fuller et al. [1983]. This method involved assigning values to the arcs of the game tree, then computing a value for each leaf by summing the values on all arcs to it from the root. In particular, the arc values were taken independently from a uniform distribution over the set $\{0,1,..,100\}$. The static values of a node were defined to be the sum of the arc values on the path from the root to the node.

C. Othello

Since othello is a widely known game, the description of its rules is skipped here. The reader is referred to [Hasagawa, 1977].

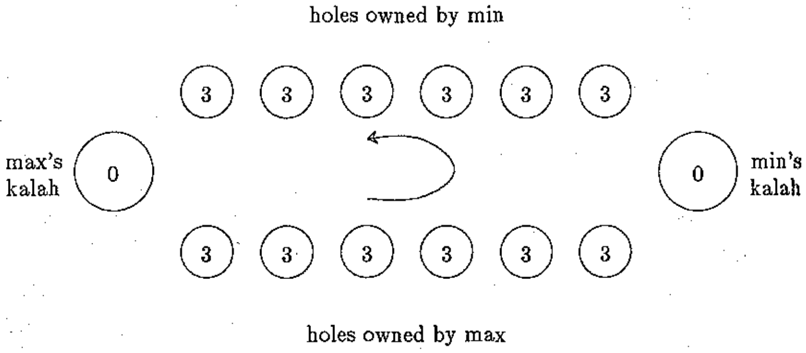

FIGURE 11: The starting position for a kalah game with six bottom holes and three stones in each hole.

D. Kalah (adapted from [Slagle & Dixon, 1969])

Figure 11 shows the starting position for three–in–a–hole kalah, and Figure 12 shows a sequence of possible moves. A player wins if he gets more than half the stones in his kalah.

To make a move, a player first picks up all the stones in one of his holes. He then proceeds counterclockwise around the board, putting one stone in each hole, including his own kalah, but skipping his opponent's kalah until all the picked–up stones are gone. What happens next depends on where the last stone lands. There are three alternatives. If the last stone lands in the player's own kalah, he makes another move. This is called a "go again." The second alternative is called a "capture." If the last stone lands in an empty hole owned by the player, then all the stones in the opponent's hole directly opposite is captured by the player. The player places all the captured stones and his own last stone in his kalah, and the opponent moves next. The third alternative is the simplest case. If the last stone lands so that neither a go–again nor a capture occurs, then the opponent moves next.

starting position	$0\begin{array}{c}333333\\333333\end{array}0$
max starts with a move	$0\begin{array}{c}333334\\3333\underset{\uparrow}{0}4\end{array}1$
min makes a go–again move	$1\begin{array}{c}44\overset{\downarrow}{0}334\\333304\end{array}1$
min goes again	$2\begin{array}{c}5\overset{\downarrow}{0}0334\\443304\end{array}1$
max makes a capture move	$2\begin{array}{c}500304\\\underset{\uparrow}{0}54404\end{array}5$

FIGURE 12: The first few moves of a kalah game, played on the board shown in Figure 11. Each time a move is made, the place from where the stones were moved is marked with an arrow (↑ or ↓).

There are two conditions which end the game. If a player gets more than half of the stones in his kalah, the game is over and he is the winner. If all the holes owned by one player, say min, become empty (even if it is not his turn to move), then all the stones remaining in max's holes are put in max's kalah and the game is over. In either case the winner is the player who has more stones in his kalah at the end of the game.

Since we considered games with no ties, we added a new rule in our simulations. If right after a move, the player acquires exactly half of the stones, then he is the winner. It is obvious that with the addition of this new rule, that the game of kalah has no ties.

REFERENCES

[Abramson, 1985]

Abramson, B., "A Cure for Pathological Behavior in Games that Use Minimax," *First Workshop on Uncertainty and Probability in AI* (1985).

[Ballard, 1983]
Ballard, B. W., "Non–Minimax Search Strategies for Minimax Trees: Theoretical Foundations and Empirical Studies," Tech. Report, Duke University, (July 1983).

[Chi & Nau, 1986]
Chi, P. and Nau, D. S., "Predicting the Performance of Minimax and Product in Game Tree Searching," *Second Workshop on Uncertainty and Probability in AI* (1986).

[Fuller et al. 1973]
Fuller, S. H., Gaschnig, J. G., Gillogly, J. J. "An analysis of the alpha–beta pruning algorithm," *Dept. of Computer Science Report, Carnegie–Mellon University* (1973).

[Hasagawa, 1977]
Hasagawa, G., *How to Win at Othello*, Jove Publications, Inc., New York (1977).

[Nau, 1980]
Nau, D. S., "Pathology on Game Trees: A Summary of Results," *Proc AAAI–80* , pp. 102–104 (1980).

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

[Nau, 1983]
Nau, D. S., "On Game Graph Structure and Its Influence on Pathology," *Internat. Jour. of Comput. and Info. Sci.* Vol. 12(6) pp. 367–383 (1983).

[Nau, Purdom, and Tzeng, 1985]
Nau, D. S., Purdom, P. W., and Tzeng, C. H., "An Evaluation of Two Alternatives to Minimax," *First Workshop on Uncertainty and Probability in AI* , (1985).

[Pearl, 1981]
Pearl, J., "Heuristic Search Theory: Survey of Recent Results," *Proc. IJCAI–81* ., pp. 554–562 (Aug. 1981).

[Pearl, 1984]
Pearl, J., *Heuristics*, Addison–Wesley, Reading, MA (1984).

[Slagle & Dixon, 1969]
Slagle, J. R. and Dixon, J. K., "Experiments with Some Programs that Search Game Trees," *JACM* Vol. 16(2) pp. 189–207 (April 1969).

[Slagle & Dixon, 1970]
Slagle, J. R. and Dixon, J. K., "Experiments with the M & N Tree–Searching Program," *CACM* Vol. 13(3) pp. 147–154

[Teague, 1985]
Teague, A. H., "Backup Rules for Game Tree Searching: A Comparative Study," Master's Thesis, University of Maryland (1985).