Product Propagation: A Backup Rule Better Than Minimaxing?

Hermann Kaindl, *Senior Member, IEEE*, Helmut Horacek, and Anton Scheucher

***Abstract*—There is a gap between theory and practice regarding the assessment of *minimaxing* versus *product propagation*. The use of minimaxing in real programs for certain two-player games like chess is more or less ubiquitous, due to the substantial search space reductions enabled by several pruning algorithms. In stark contrast, some theoretical work supported the view that product propagation could be a viable alternative, or even superior on theoretical grounds. In fact, these rules have different conceptual problems. While minimaxing treats heuristic values as true values, product propagation interprets them as independent probabilities. So, which is the better rule for backing up heuristic values in game trees, and under which circumstances? We present a systematic analysis and results of simulation studies that compare these backup rules in synthetic trees with properties found in certain real game trees, for a variety of situations with characteristic properties. Our results show yet unobserved complementary strengths in their respective capabilities, depending on the size of node score changes ("quiet" versus "nonquiet" positions), and on the degree of advantage of any player over the opponent. In particular, exhaustive analyses for shallow depths show that product propagation can indeed be better than minimaxing when both approaches search to the same depth, especially for making decisions from a huge amount of alternatives, where deep searches are still prohibitive. However, our results also provide some justification for the more or less ubiquitous use of minimaxing in chess programs, where deep searches prevail and the pruning algorithms available for minimaxing make the difference.**

***Index Terms*—Game trees, minimaxing, multivalued evaluation functions, product propagation, simulation studies.**

I. Background and Introduction

While the game of checkers has been solved [1], there are many other interesting games that are not, and some will never be. That is, except for rare positions of such a game, there is no practical way of determining the exact status (the *true* value) of non-goal nodes that represent most of the positions. Therefore, it is usually necessary to resort to *heuristic* estimates of their "goodness" or "strength" for one side. Such values are assigned by a so-called *static evaluation function* f which incorporates heuristic knowledge about the domain in question. In this paper it is sufficient to consider $f(n)$ as some function that evaluates each node n (that represents a position) with some error.

Given such an evaluator, the question remains how its values can be used for making reasonable decisions. Since the immediate application of f to the children of the given node usually does not lead to good decisions in practice, it seems "natural" to look ahead by searching deeper and evaluating the resulting nodes. For such a procedure, it remains to be specified how deep the various branches should be searched (to uniform or variable depth [2], [3]) and how the heuristic values should be backed up (i.e., propagated) from the given node's children.

While the backup quality through two different rules is the focus of this paper, we decided to stick with searches to uniform depth, since this allows a fair comparison through exactly the same number of backups for both rules throughout. Although in practice searches are to variable depth, we can compare the two backup rules using strictly uniform depths without losing generality. In fact, a heuristic evaluation at some depth can, in principle come from a deeper search below, but it is sufficient for the purposes of our study that it evaluates with a certain quality. The latter is characterized here with an error probability, and it is, of course, the same for both backup rules. So, any possible effect of variable depth of search trees related to the compared backup rules is outside the scope of this paper.

A. Minimaxing

In *two-person games* with *perfect information*, the most successful approach for backing up values in practice has been *minimaxing* (for a description see, e.g., [4]). In the following, we assume that $f(n)$ assigns a value to a node n from the viewpoint of the moving side at n. Since the *depth* of such searches is important for the purpose of this paper, we also define here a special case where all branches are searched to the same depth.

Definition 1: A *minimax value* $MM_f(n)$ of a node n can be computed recursively as follows (in the *negamax* formulation complementing the successor values, in order to avoid handling the two cases of maximizing and minimizing separately).

1) If n is considered *terminal*: $MM_f(n) \leftarrow f(n)$.
2) else: $MM_f(n) \leftarrow \max_i(-MM_f(n_i))$ for all child nodes n_i of n.

While this formulation would also cover variable-depth search, $MM_f^d(n)$ is the minimax value of node n resulting from exactly d applications of the recursion (2) in every branch of the search tree. That is, $MM_f^d(n)$ defines the minimax value from a *full-width search* of the subtree rooted at n to a uniform depth d.

Manuscript received November 05, 2014; revised October 06, 2015; accepted December 07, 2015. Date of publication December 17, 2015; date of current version June 14, 2017.

H. Kaindl is with the Institute of Computer Technology, TU Wien, Vienna, Austria (e-mail: kaindl@ict.tuwien.ac.at).

H. Horacek is with the German Research Center for AI, Saarbrücken, Germany (e-mail: helmut.horacek@dfki.de).

A. Scheucher is with the Raiffeisen Bank International, Vienna, Austria (e-mail: anton.scheucher@gmx.at).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2015.2508966

The use of minimaxing in computer chess practice is more or less ubiquitous. For instance, the special chess machine Deep Blue, which defeated the highest-rated human chess player for the first time in a match consisting of several games under tournament conditions, used minimaxing. Current chess programs still use minimaxing, but with much more variable search based on so-called null moves, see, e.g., [3]. As explained above, variable search depth is outside the scope of our paper.

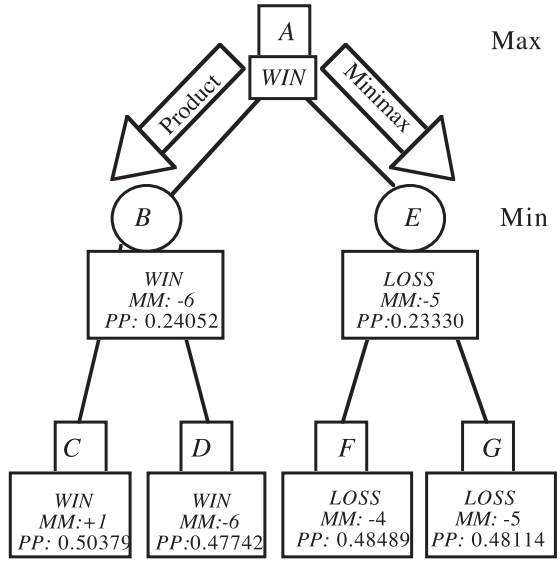
Yet there has been some theoretical doubt on the usefulness of minimaxing as pointed out by Pearl [5]. In fact, minimaxing treats heuristic estimates as if they were true values. Pearl compared this to committing one of the deadly sins of statistics, computing a function of the estimates instead of an estimate of the function. So, there is some lack of theoretical foundation and explanation for the (very successful) use of minimaxing in game-playing practice.

Even to the contrary, Nau [6] and Beal [7] showed independently that for certain classes of game trees the decision quality is degraded by searching deeper and backing up heuristic values using the minimax propagation rule. Nau called such behavior *pathological* (see also [5], [8] for successor work).

More realistic results about the effects of deeper and deeper searches using minimaxing were achieved through an investigation under more realistic game tree conditions when using *multivalued* evaluation functions [9]. Such functions have many distinct values as their result and can discriminate between positions according to the heuristic knowledge represented in these values. Simulation studies of this multivalued model have exhibited sharp error reductions for deeper searches using minimaxing, as observed in practice. The error reductions are primarily due to the *improved evaluation quality* as search depth increases, although the same evaluation function is used at all levels of the tree, and although its general error probability is independent of the depth. Essentially, with increasing search depth, the evaluation function is more frequently used on such positions that can be more reliably evaluated by a multivalued function with the assumed properties, which are also found in practice. This effect together with the ability to discriminate between positions of different "goodness" leads to the benefits of using *multivalued* evaluation functions (of appropriate granularity) for minimaxing. While these results are more general than those from experiments using concrete game-playing programs, they have a close relation to, e.g., computer chess and checkers practice. A more general investigation of deeper look-ahead using minimaxing appeared more recently in [10].

B. Product Propagation

Since the benefits of using minimaxing in practice had not been explained theoretically for a long time, different backup rules have been proposed, such as *product propagation* [5] (in fact, this rule was already used much earlier by Slagle and Bursky [11]). It requires that an evaluation function $f'(n)$ returns values between 0 and 1 that are estimates of the probability that the position represented by node n is a forced win. $f'(n)$ assigns a value to a node n from the viewpoint of the moving side at n.



WIN, $LOSS$ True value
MM Minimax back-up value or heuristic value of terminal node
PP Product back-up value or probability to win of terminal node

Fig. 1. Move decision error by Minimax.

Definition 2: A *probability estimate* $PP_{f'}(n)$ of a node n can be computed recursively as follows (in the *negamax* formulation).

1) If n is considered *terminal*: $PP_{f'}(n) \leftarrow f'(n)$.
2) else: $PP_{f'}(n) \leftarrow 1 - \prod_i (PP_{f'}(n_i))$ for all child nodes n_i of n.

$PP^d_{f'}(n)$ is the depth d estimate of node n resulting from exactly d applications of the recursion (2) in every branch of the search tree, in analogy to $MM^d_f(n)$ as defined above.

Product propagation is theoretically sound for *independent* probabilities. However, as noted already by Slagle and Bursky such probabilities are generally *not* independent in practice.

C. Two Representative Examples of Errors Made

In order to get a better understanding of the relative qualities of these backup rules, let us have a look at representative examples of move-decision errors. For ease of presentation, we select examples with the shallowest search depth of 2 that make a difference to the move decision of Minimax and Product, respectively. "Minimax" and "Product" are players that use minimaxing (see Definition 1) and product propagation (see Definition 2), respectively.

Each node (A to G in Figs. 1 and 2) is associated with its *true value* (WIN or $LOSS$), a Minimax score MM according to the function $f(n)$, and a Product score PP according to function $f'(n)$, where both $f(n)$ and $f'(n)$ provide heuristic estimates of the true value of node n. For a fair comparison, $f(n)$ and $f'(n)$ have to correspond, of course. Below we define exactly how $f'(n)$ is derived from $f(n)$. The value of the evaluation function $f(n)$ (and its derived $f'(n)$) is considered erroneous for position n if its preference is inconsistent with the true value, that is, $f(n) < 0$ and $f'(n) < 0.5$ for a

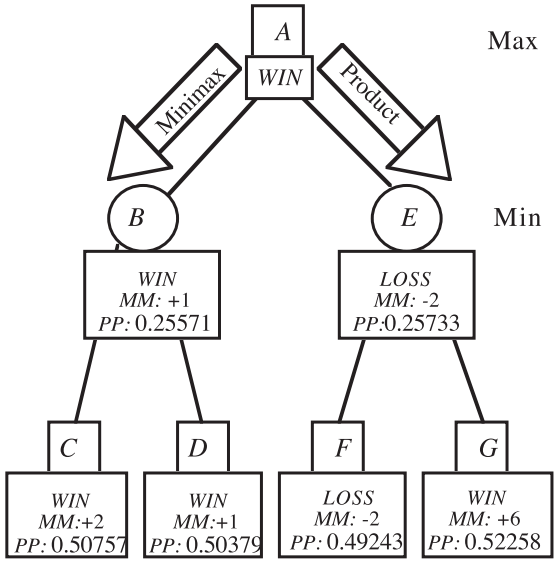


Fig. 2. Move decision error by Product.

won position, as well as $f(n) > 0$ and $f'(n) > 0.5$ for a lost position, respectively.

In the example of Fig. 1,[1] Minimax takes the wrong move to a losing position E because of the gross evaluation error of position D in the other subtree. The wrong heuristic estimate is treated by Minimax as if it were a true value. Product compensates for this evaluation error in this example through taking into account the evaluation of position C as well.

In the example of Fig. 2, Product takes the wrong move to a losing position E. In essence, the high probability to win estimated for position G more than outweighs the much smaller probability to win estimated for position F (which amounts even to a small estimated probability to lose of 0.50757), compared to small probabilities to win estimated for both positions C and D in the other subtree. In contrast, Minimax makes the correct decision in this example by avoiding the position F due to its negative evaluation. These examples are consistent with the major result of [12], [13], which relates the respective advantages of minimaxing and product propagation to the strength of the evaluation function. While larger errors of this function favor product propagation, smaller ones favor minimaxing. The example of Fig. 1 contains a gross evaluation error of a terminal node, while there is even *no* such error in Fig. 2.

D. Different Views on These Propagation Rules

However, these examples suggest a different view of these backup rules than the one expressed by Pearl in [4], [5]. Pearl states that minimaxing may be useful for beating a fallible opponent, since this backup rule contains a realistic model of a fallible opponent who shares the assessment of the terminal values by the player Minimax—such an opponent can be *predicted* to choose the moves evaluated best in the search tree of Minimax.

Actually, it depends on the errors made by the evaluation function. Under the conditions in our model[2] (and presumably in real computer chess programs), it is Product who would play for the option of taking advantage of an error by a fallible opponent as illustrated in the example of Fig. 2. As explained above, Product decides for the wrong move to the lost position E, where a strong opponent can be expected to move to position F for exploiting this error made by Product. A fallible opponent, however, may commit a blunder by moving to position G, which is better for the player Product than any of the two positions C or D, which can result after the correct move to position B (instead of E) before. As shown with this example, Product can be viewed to 'hope' for a blunder by a fallible opponent. So, this is in contrast to the view expressed by Pearl [4], [5].

Some authors (such as [5], [12], [14], [15]) conjecture that product propagation could be a viable alternative to minimaxing. So, this theory and, e.g., computer chess practice seem to be in conflict. Therefore, we compared the performance of product propagation to that of minimaxing in several ways. For depth 2, we carried out a *systematic analysis* of the differences between the backup rules according to combinations of evaluation errors. In effect, this is a *complete* analysis of their respective decision quality and, therefore, its associated results are without uncertainty (it was published under this heading in our related conference paper [16]). For depth 3, we made comprehensive and systematic tree searches in the game tree model introduced in [9], since it appears to capture most closely the essential conditions encountered in game-playing practice. For less shallow search depths, we made *simulation studies* in this game tree model with statistically significant results.

E. Paper Overview

The remainder of this paper is organized in the following manner. First, we summarize previous work on comparing minimaxing with product propagation. In order to make this paper self-contained, we also explain the tree model introduced in [9]. Then we present a *qualitative* analysis of the backup differences between minimaxing and product propagation (based on the study of their respective decision quality in [16]). Since this exhaustive analysis is practically restricted to depth 2 search, we subsequently present a *quantitative* analysis of systematic tree searches to depths 2 and 3. For an investigation of less shallow search depths, we present a simulation study, which includes game contests in simulated trees.

II. Previous Work

While there has been a fair bit of attention on the theoretical problem of minimaxing pathology, some of which we have mentioned in the introduction, we focus here on the work that directly compares minimaxing with product propagation under various conditions.

Nau [14] investigated product propagation as an alternative to minimaxing in games where the values of the real leaf nodes

[1] In the presentation of such an example, it is better to use the minimax instead of the negamax formulation for pedagogical reasons. Therefore, we show all values from the viewpoint of the moving side at A.

[2] For details about the underlying game tree model, see Section III.

directly correspond to the values of the squares in the initial board configuration, which are randomly assigned one of two values independently of the values of the other squares. Under these conditions, Nau's experiments resulted in a higher probability of correct move decision using product propagation compared to minimaxing. In a game contest, a program based on product propagation scored marginally better than an otherwise identical program based on minimaxing. In similar games, but with incremental dependencies of true game-theoretic values, the results showed about the same probability of correct move decision for both backup rules.

In later work, Nau et al. [15] reported that product propagation scored better than minimaxing (at most search depths) in a game contest with independent probabilities, when counting the *critical games* only.[3] Further experiments showed, however, that minimaxing was better than product propagation (for search depths 3 and 4) in a game contest with incremental dependencies.

Nau [13] defined games with dependencies of true game-theoretic values in graphs where sibling nodes have many children in common. A comparison of minimaxing and product propagation on these games indicated some influence of the evaluation function used. Game contests revealed that product propagation performed better than minimaxing if some evaluation function was used and worse than minimaxing if another function was used that is more accurate for these games. Results by Chi and Nau [12] confirmed this relationship of the respective advantages of these rules to the strength of an evaluation function used: the stronger the evaluation function the better for minimaxing.

Additionally, Chi and Nau compared these backup rules on several games, including a small variant of kalah. Most interestingly, in this real game a program based on product propagation performed better than its opponent based on minimaxing.

Since both programs searched to the *same depth*, however, these comparisons were unfair for minimaxing, which could have utilized well-known pruning procedures for searching much deeper with the same number of nodes generated. (For a comparison of pruning procedures see [17].) Still, there was some indication that product propagation may be the better rule for backing up heuristic values.

Nau et al. [15] as well as Baum [18] investigated combinations of minimaxing and product propagation. Their results suggested that the respective advantages could be combined by some combination of these backup rules.

In summary, the previous theoretical work on comparing minimaxing with product propagation was not conclusively in favor of either backup rule. In particular, it neither provided an explanation why product propagation is neglected, e.g., in computer chess practice, nor whether this would be justified.

Note that all this related work above appeared a long time ago, so it seems as though this topic has been laid to rest. Even the more recent investigation of the effects of deeper look-ahead mentioned above focuses on minimaxing only and does not include product propagation [10]. Interestingly, another recent study proposed a new propagation rule named "error minimizing minimax" bearing some resemblance to the product propagation rule [8]. However, it requires both heuristic values *and* error estimates on those values, where the latter are rarely available for, e.g., chess programs. Therefore, we consider this approach outside the scope of our paper and rather provide new results for the comparison of minimaxing and product propagation per se.

Other, more recent research employs a distribution calculated through probability propagation for guiding best-first search in the context of proving the outcome of an adversarial game (in the sense of its *true value* such as WIN or $LOSS$) [19], [20]. Like Proof Number search [21], a best-first search algorithm for finding solutions to problems represented in AND/OR trees, this addresses efficiency of search rather than quality of backup values in bounded search.

Yet other recent research on very selective search in computer Go employs Monte Carlo Tree Search (see, e.g., [22]), a best-first tree search algorithm that evaluates each state by the average result of simulations from that state. It has recently been combined with minimaxing [23].

All this more recent work does not, however, address or answer the question that we take up again in this paper, whether product propagation is a better backup rule than minimaxing.

III. The Underlying Game-tree Model and Tree Generation

The underlying game-tree model is the same as the one introduced in [9]. In order to make our presentation self-contained, we briefly sketch it here.

The following assumptions of the underlying game-tree model are derived from Pearl's [5] basic model (as given in [9]):

1) the tree structure has a uniform branching degree b;
2) true values of nodes (TV) are either WIN or $LOSS$;
3) true values have the game-theoretic relationship of two-person zero-sum games with perfect information[4];
4) heuristic values h [assigned to a node n by a static evaluation function $f(n)$] are elements of the set $\{-h_{\max}, \ldots, -1, +1, \ldots, +h_{\max}\}$.[5]

For product propagation, a function f' is needed, which estimates probabilities to win. f' is functionally dependent on f as given in Section III-A, where the following monotony conditions are required:

1) $f(n) = 0$ implies $f'(n) = 0.5$[6];
2) $f(n_1) > f(n_2)$ implies $f'(n_1) > f'(n_2)$.

[3]For each initial game board, one game was played with one player moving first and another was played with his opponent moving first. For some game boards, one player was able to win both games of the pair. These are called critical games.

[4]A nonterminal node is won for the side to move if at least one of its child nodes is won, and it is lost otherwise.

[5]In chess, a simple evaluation function which is based on the sum of the chess pieces' material values gives rise to a value of 42 for $h_{\max}$. Discussions about finer-grained evaluation functions can be found in [9]. However, the qualitative effects are the same, and restricting to the material balance avoids that the reader must be very familiar with specifics of chess. So, for showing the major effects in this paper we use $h_{\max} = 42$.

[6]$f(n) = 0$ actually cannot occur for the given definition above taken from [9], which avoids intricacies with error definitions because the game model does not contain a draw. For expressing monotony constraints, the concern about complication in error analysis is irrelevant.

Two additional conditions are taken from [9], where they are derived and their rationale is given.

A. Non-uniformity of Error Distribution

Whenever a heuristic value incorrectly estimates the true value, an error occurs. We characterize the error made by an evaluation function f as a probability of error. The nonuniform error distribution in this model

$$e_c(h) = \begin{cases} w_c(h) & h < 0 \\ 1 - w_c(h) & h > 0 \end{cases} \tag{1}$$

is based on the following distribution of the *probability to win* (for some constant c):

$$w_c(h) = \begin{cases} \frac{1}{2} + \frac{1}{2\arctan(c)} \arctan(c\frac{h}{h_{\max}}) & \text{if } h = f(n) \in [-h_{\max}, h_{\max}] \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

This function was developed from a simpler one proposed by Pearl [4, p. 360] (also based on the arc tangent function). Horacek [24] informally used a function with similar shape in order to handle reasoning with uncertainty in a computer chess program. Chess and checkers programs, for instance, assign large heuristic values to positions evaluated as clearly favorable. In practice, they win games more often when they achieve such positions on the board. The higher these values, the more likely the program is winning. Therefore, it is clearly reasonable to assume that small heuristic values have a low probability to win, which increases monotonically with increasing heuristic values. Furthermore, for positive heuristic values it is more likely to win than to lose.

Since Pearl's function based on the arc tangent cannot be parameterized, (2) generalizes it to a set of "probability to win" distributions $w_c(h)$ through its parameter c.[7] It allows specifying the slope of the curve. (For simplicity, we assume a continuous range of heuristic values at this point.)

The actual mapping from f to f' makes use of this function as follows: $f'(n) = w_c(f(n))$. So, the monotony conditions required are fulfilled.

B. Dependency of Heuristic Values

The heuristic values of child nodes depend on the heuristic value of the parent node due to the incremental dependency defined as follows. If 0 is the minimum change and a denotes the maximum change in the evaluation between node n and its child nodes n_i (for all i from $1..s$, with s being the number of child nodes), the following constraint holds:

$$\begin{aligned} MAX:&\ f(n) \le f(n_i) \le f(n) + a \\ MIN:&\ f(n) - a \le f(n_i) \le f(n) \end{aligned} \tag{3}$$

[7] $\int_{-\infty}^{\infty} w_c(h)dh = 1$

C. Tree Generation in General

The basic scheme in this model is that the side to move can improve or at least maintain its score by making a move in a few distinct ways, each of which is associated with some (positive or zero) score increment.[8] According to this scheme, binary game trees (with exactly two child nodes for each non-leaf node) of uniform depth d_g are generated top-down, beginning at the root node by a recursive procedure. Starting with a heuristic value assigned to the root node, heuristic values are assigned to the child nodes n_i of a node n such that the heuristic values are established.[9] Once given the heuristic values, the true values can be assigned to the nodes n_i with the probability of error e_c under the constraint of the game-theoretic relationship.[10]

Based on this general approach to tree generation, our different studies require specific enhancements. For the exhaustive analyses, complete enumeration is required, of course. For the simulation studies, in contrast, stochastic events are simulated. More details are given in the respective sections below.

IV. Qualitative Analysis of Backup Differences

First, we analyze differences between the competing backup rules in purely qualitative terms, that is, independently of assumptions about the concrete value distribution in the game tree. In this setting, we clearly cannot expect a general finding that one of the backup rules is outperforming the other. We do, however, intend to identify influence factors which contribute to superior performance of Minimax or Product, respectively. We have to restrict this exhaustive analysis to the case of depth 2 searches for reasons of complexity explained below in more detail. In effect, this is a comparative analysis of the *decision quality* of Minimax vs. Product.

A. The Basic Situation for a Comparison

We carried out a systematic analysis of the differences between the backup rules according to combinations of evaluation errors. The basic situation in which differences between minimaxing and product propagation manifest themselves is illustrated in Fig. 3, for depth 2 and branching degree 2, with node labels as indicated. In order for this diagram to represent a *critical case*, two conditions concerning the values associated with the nodes must hold:

1) the backup rules must select different moves;
2) one move leads to a won position, while the other leads to a lost position.

Whether the first condition holds can be derived from the definitions of the competing backup rules. Without loss of generality (due to symmetries), we assume that Max is on move in the root position R, M the node preferred by Minimax, P the one preferred by Product, and that $f'(M_1) \ge f'(M_2)$ and

[8] This does obviously not include *Zugzwang* cases as in chess, but they occur very rarely, so that we exclude them from our modeling approach.

[9] According to the tree model, uniformly distributed integer increments ranging from 0 to 8 are assumed, using $a = 8$ in Eq. (3).

[10] Due to the shape of the function e_c (see (1)), heuristic values must be assigned to nodes n_i *before* the true values, because it is impossible to achieve such an error distribution starting with the true values WIN and $LOSS$.

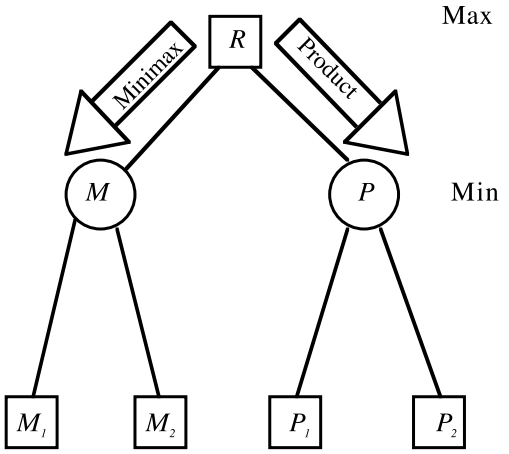


Fig. 3. Basic situation with differences between Minimax and Product.

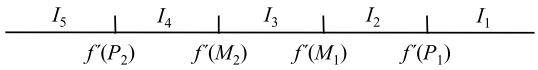


Fig. 4. Intervals of the draw value.

$f'(P_1) \geq f'(P_2)$ hold.[11] Under these conditions, $f'(M_1)$ and $f'(M_2)$ must both be greater than the value of the smaller of the successor nodes of P, $f'(P_2)$, according to minimaxing. Conversely, product propagation demands that the product of the successors of P is greater than the product of the successors of M. Therefore, the value of the larger of the successor nodes of P, $f'(P_1)$, must be greater than both $f'(M_1)$ and $f'(M_2)$. Consequently, there exists a partial ordering between the values of these four nodes

$$f'(P_2) \leq f'(M_2) \leq f'(M_1) \leq f'(P_1). \tag{4}$$

The second property of a critical case depends on the true values associated with the competing nodes, M and P, which in turn depend on the true values of their successor positions. Hence, we have to distinguish which of the positions M_1, M_2, P_1, and P_2 are evaluated correctly and which ones are not. This yields 16 cases to be considered.

In our analysis, we also distinguish how the transition point between loss and win (the *draw value*, 0 in terms of f, and 0.5 in terms of f') relates to these four values. This yields five cases, leading to a total of 80 cases. Among these cases, those are relevant where the move decision matters, that is, where M is won and P lost for the side to move, or vice versa. Together with the partial ordering on the evaluations of the positions under discussion, the draw value can be in one of the intervals as labeled in Fig. 4. This information helps us to distinguish the various cases of value assignments of the leaf nodes. For example, if the draw value is in the interval I_2, then the true value of $P_2 = WIN$, while all the other three nodes have the value $LOSS$.

[11]Because of the monotony condition, it does not matter that Minimax actually works with f rather than f', where the former can be mapped to the latter. We cannot do this ordering with f, however, since a product of probabilities is involved, as given below.

B. Properties of the Basic Situation

For illustrative purposes, let us discuss a representative case of the basic situation. If there is no evaluation error, then only the case where the draw value falls in interval I_4 makes a difference between the competing backup rules, since M is won and P lost in I_4 (compare with the example in Fig. 2). If the draw value falls in interval I_5, then all values are 'positive' (that is, $f > 0$, i.e., $f' > 0.5$) and both M and P are won. If the draw value falls in one of the intervals I_3, I_2, or I_1, then both M and P are lost.

The analysis of all 80 cases along similar lines yields the differences between minimaxing and product propagation summarized in Table I, where only the critical cases have to be dealt with. The left part contains the constellations where minimaxing makes the right decision and product propagation does not, and the right one those where only product propagation succeeds. The lines are ordered according to the number of evaluation errors associated with the four positions, and in each cell, the erroneous positions are indicated, together with the interval in which the draw value falls in each case.

Despite the high degree of abstraction of the information contained in the table entries, it shows that advantages and disadvantages of minimaxing and product propagation are balanced in terms of the number of cases. However, this does not necessarily imply that both propagation rules are equally good in general. Their relative move decision quality depends on further conditions as analyzed in detail in [16] and summarized below.

C. Analysis with a Simplifying Assumption

The comparison leads to an interesting result for the special case that the heuristic values have a *uniform error distribution*. Based on this simplifying assumption, only the two extreme cases favoring minimaxing and the two average cases favoring product propagation, as well as the difference between the number of cases falling in either of the intervals I_2 and I_4 remain as decisive factors, whatever the distribution of values in a game tree is. For the trivial case of board evaluator competence of exactly 50%, that is, guessing, these cases are leveled out completely, For the case of constant error probabilities, the comparison between product propagation and minimaxing can be nailed down analytically to a simple difference, which makes the compensative factors favoring the competing backup rules evident.

By putting together all the cases listed in Table I, substantial simplifications can be made. To start with, each pair of cases, one favoring minimaxing and the other favoring product propagation, both being associated with the same number of errors and falling in the same interval, cancels itself out. For example, there are two cases with a single error in interval I_3—see the third line below the header in Table I—one favoring minimaxing (the error occurs in position M_2), the other favoring product propagation (the error occurs in position P_2).

This leaves us with only eight remaining cases, half of which related to interval I_2, the other half related to interval I_4. Each case comprises four positions, each of which may be

TABLE I
SUMMARY OF PREFERENCES (OF THE CRITICAL CASES) BASED ON COMPLETE ANALYSIS OF THE BASIC SITUATION WITH DIFFERENCES BETWEEN MINIMAX AND PRODUCT

	intervals favoring	
	minimaxing	product propagation
no error	I_4	
one error (position)	$I_4(P_1)$	$I_2(P_2)$
	$I_3(M_2)$	$I_3(P_2)$
	$I_5(P_1), I_5(P_2)$	$I_5(M_1), I_5(M_2)$
two errors (positions)	$I_1(M_1, M_2)$	$I_5(M_1, M_2)$
	$I_2(M_1, M_2)$	$I_2(M_1, P_2), I_2(M_2, P_2)$
	$I_3(P_1, M_2)$	$I_3(M_1, P_2)$
	$I_4(P_1, P_2)$	$I_4(M_1, P_2), I_4(M_2, P_2)$
	$I_5(P_1, P_2)$	$I_1(P_1, P_2)$
three errors (positions)	$I_2(M_1, M_2, P_1)$	$I_4(M_1, M_2, P_2)$
	$I_3(M_2, P_1, P_2)$	$I_3(M_1, M_2, P_2)$
	$I_1(M_1, M_2, P_2), I_1(M_1, M_2, P_1)$	$I_1(M_2, P_1, P_2), I_1(M_1, P_1, P_2)$
four errors	I_2	

erroneously evaluated (with probability p) or correctly (with probability $1-p$). The total frequency for a case amounts then to $p^n*(1-p)^{(4-n)}$ for exactly n positions being erroneously evaluated.

A calculation of the compensative benefits of the competing backup rules, separately done for intervals I_4 and I_2 looks as follows. Positive values are in favor of minimaxing, negative ones in favor of product propagation in formula (5), and vice versa in (6), as follows:

$$I_4 : ((1-p)^4+(1-p)^3*p-(1-p)^2*p^2-(1-p)*p^3) \quad (5)$$

$$I_2 : ((1-p)^3*p+(1-p)^2*p^2-(1-p)*p^3-p^4) \quad (6)$$

For details of the derivation, see [16].

The expressions in formulas (5) and (6) both contain a common factor $g(p) = (1-p)^3 + (1-p)^2*p - (1-p)*p^2 - p^3$, so that (5) can be rewritten as $(1-p)*g(p)$ and (6) as $p*g(p)$.

Let $N(I_4)$ and $N(I_2)$ be the numbers of positions falling in intervals I_4 and I_2, respectively. If

$$I_2, I_4 : (1-p)*N(I_4) > p*N(I_2) \quad (7)$$

holds, minimaxing is superior to product propagation, otherwise the opposite is the case.

Apparently, the comparison depends on two complementary factors—the error probability, and the relation between the number of cases falling in intervals I_2 and I_4. Assessing the impact of the error probability is simple—the better the board evaluator is, the better this is for minimaxing. Assessing the impact of the relation between the number of cases falling in intervals I_2 and I_4 involves the following argument. In order for Product to make a move decision different from that of Minimax, the interval I_2 must be larger than I_4, otherwise $f'(P_1)*f'(P_2)$ is not greater than $f'(M_1)*f'(M_2)$. Consequently, there are more cases in which the draw value falls in interval I_2 than this is the case for interval I_4. This means that product propagation is superior to minimaxing for evaluators that are not 'too good', that is, for evaluators where p is not small enough so that $(1-p)/p \leq N(I_2)/N(I_4)$ [cf. (7)].

As shown below, it requires an evaluator with at most a few percent error rate to make minimaxing superior to product propagation for the kind of constellations considered here (that is, constant error probabilities and the tree model underlying the analyses below).

The model as it has been discussed so far, accounts for depth 2 searches only, since it exploits a particular coincidence that is only present for these shallow searches. The leaf node values of product propagation and minimaxing correspond exactly with $f'(n) = w_c(f(n))$. This is only the case for static values, but not for backup values returned from deeper searches, even though they may derive from corresponding evaluations of their leaf nodes. An attempt to extend the model to larger search depths would cause a combinatorial explosion of the number of cases to be distinguished. This is mainly because the strict ordering between the values P_2, M_2, M_1, and P_1 (see Fig. 4) does not hold for P_2 in depth 3 for dynamic values, with the following consequences.

- There are eight values (four Minimax and four Product values) to consider instead of four, each of them requiring a distinction of whether it is correct or not (256 cases instead of only 16).
- There are nine possible positions instead of five for the draw value.
- Unlike for the basic case of depth 2 searches, variations in the relative order of the values associated with the nodes at depth 2 must be distinguished for depth 3 searches, especially for the product propagation values. The combinations here are less regular, so we did not do precise calculations of the number of cases.

Altogether, combinatorics yields in the range of 80 000 cases for a depth 3 analysis as opposed to the 80 cases for depth 2, which makes it impractical to pursue the analysis along the same lines. So, our qualitative analysis essentially boils down to a comparative analysis of the *decision quality* of Minimax vs. Product. Both search to a fixed depth of exactly 2, where they

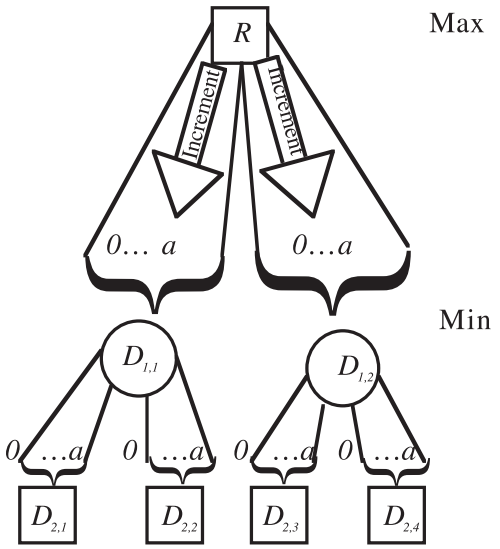


Fig. 5. Illustration of complete tree generation.

backup from exactly corresponding static values, and apply their respective backup rules exactly once. The major result is that Product's decisions are slightly more often correct than the ones of Minimax.

V. Quantitative Analysis of Backup Preferences

In order to compare the two competing backup rules also on trees of depth 3 in terms of critical cases, we perform a quantitative analysis of backup preferences. We compute the outcome of all possible search trees according to a specific tree model, the one defined in [9] and summarized above. In these search trees, we accumulate the respective errors made by the two backup rules in critical cases.

First, we explain the complete tree generation with full enumeration. Based on that, we present our quantitative analyses and results for depths 2 and 3.

A. Complete Tree Generation

Based on the general tree generation approach explained above, let us elaborate on the specifics of the complete tree generation with full enumeration (see Fig. 5 for an illustration). Depending on the investigated case, some heuristic value is assigned to the root node R. Each search tree is built recursively as indicated above. Through full enumeration of all increments at each node, the complete set of trees is created.

Explaining this in more detail for depth 1 first, a representative tree with node $D_{1,1}$ is created with some increment, e.g., 0, and analogously node $D_{1,2}$ with some independent increment. This is repeated for all possible increments for both branches. According to the general tree generation approach explained above, uniformly distributed integer increments ranging from 0 to 8 are assumed, using $a = 8$ in (3). Since we generate binary trees, pairs of all possible combinations of values result, i.e., a total of 9^2 nodes.

For depth 2, the same generation procedure is applied, but because of the recursive tree generation, it has to be done for both $D_{1,1}$ and $D_{1,2}$ at level 1 as the new roots (with their respective heuristic values), and more precisely, because of the full enumeration, for each pair of instances of $D_{1,1}$ and $D_{1,2}$. So, each tree representative has leaf nodes $D_{2,1}$ to $D_{2,4}$, as shown in Fig. 5. Because of the full enumeration, a total of $N_2 = 9^6$ leaf nodes at depth 2 result, in principle (for all cases and six nodes with assigned values).

In order to cope with the combinatorics, two measures for reducing the generation effort were implemented. The first one is possible already for depth 1, since it makes no difference, whether the concrete increments leading to $D_{1,1}$ and $D_{1,2}$, are, e.g., 4 and 5 or 5 and 4, respectively. Therefore, looping over increment values is nested in such a way that each value combination is generated only once. However, it has to be counted twice for depth 1, more generally for deeper trees as often as it would occur. The second measure handles directed acyclic graphs (DAGs) instead of trees in situations where the same sum of different contributions of increments results, e.g., for increments 5 and 3 in one branch of a depth 2 tree, and 6 and 4 in another, since $5 - 3$ and $6 - 4$ have the same result.[12]

These reductions of the generation effort are, of course, even more important for the complete tree generation at depth 3. Still, when expanding the tree computations from level 2 to level 3, combinatorics in the number of search trees generated led us to focus on certain properties of interest. For $a = 8$, 9 is the number of possible cases of increments in one move in the tree model. An upper bound for the number of depth 2 trees is N_2 (ignoring the reductions above). Each depth 2 tree has 4 leaf nodes, each of which has 2 successors at level 3. Therefore, 8 new nodes result for the expansion of each tree from depth 2 to 3, so that there are $N_3 = (a+1)^8 * N_2$ depth 3 trees, that is, in the order of 10^{13}. So, combinatorics strikes already. Consequently, we reduced the value of the maximum increment a at levels 2 and 3, so that some combinations of values for a at these levels were investigated, as detailed below.

It remains to be stated exactly which errors e_c [as defined in (1)] we used in the course of this complete tree generation—$c = 4$. For estimating probabilities by Product through f', we mapped using $f'(n) = w_1(f(n))$, i.e., $c = 1$. For this special case, (2) is the same function as originally proposed by Pearl [4] to "translate" heuristic evaluations into estimates of the probability of winning the game from these positions. By using this f' function for heuristic evaluation, Product makes its move decisions, recursively backing them up to depth 1.

In the course of the complete tree generation, also the move decision errors made by Minimax and Product, respectively, are accumulated. Examples of such errors are given above, which can only occur in *critical cases*. So, these cases have to be identified, and the move decision errors accumulated. This requires backing-up the heuristic values with minimaxing and product propagation, respectively. Actually, only the backup values at depth 1 are relevant, since the move decisions are based on them. According to the first condition for a critical case, Minimax and Product must select a different move. According to the second condition, one move leads to a won position, while the other leads to a lost position. The first condition is

[12]For a MAX node, the increment is added, for a MIN node subtracted.

easy to evaluate, and only if it is fulfilled, the second one is evaluated. However, it can only be evaluated probabilistically in the course of this tree generation, since all cases have to be included, which occur with different probabilities according to (2).

Let us explain this probabilistic evaluation using the example of a move decision error as illustrated in Fig. 1. Instead of the concrete WIN and $LOSS$ assignments, however, the probabilities of WIN and $LOSS$ have to be determined for the assigned heuristic values using (2). For the left branch rooted in node B, these probabilities can simply be multiplied, since they are independent of each other. And since a WIN for node B can only occur if both successor nodes are WIN, this is sufficient. In the right branch rooted in node E, in contrast, all cases have to be treated where at least one of the successor nodes is a $LOSS$, since E is a $LOSS$. All these probabilities relevant in this case have to be multiplied again and result in the probability that Minimax makes this move decision error. For all the generated trees, these errors of Minimax and Product, respectively, are accumulated separately.

B. Quantitative Results for Depth 2

Before we elaborate on various aspects of depth 3 searches in detail, we first present results for depth 2. For the chosen search tree parameters, 3,886 critical cases result at level 2. For these, subtracting the respective error accumulation of Minimax (50.7895%) from the one of Product (49.2105%) results in a difference of -1.579 percent. Since Minimax made more errors, there is a slight preference of Product over Minimax for depth 2 searches.

For depth 2 (only), applying the oversimplifying assumption of a *uniform error distribution* enables us to compute a numerical relation between the quality of the evaluation function and the ratio between the number of cases where the draw value falls in intervals I_2 or I_4, respectively. In the qualitative analysis in the preceding section, these have been identified as the two compensative factors responsible for the superiority of Minimax or Product, respectively. Based on the assumption of a uniform error distribution, we successively increased the quality of the evaluation function, in order to determine when Minimax becomes superior to Product. This occurred at about 95.2% correct evaluations (and thus at about 4.8% incorrect ones).

Formula (7) above expresses superiority of the competing backup rules under simplified conditions for depth 2 searches. This expression, $(1-p)*N(I_4) > p*N(I_2)$, relates the error probability p to the number of cases where the draw value falls in intervals I_2 and I_4, respectively. For the score of 95.2% correct evaluations, the number of cases where the draw value falls in interval I_2, $N(I_2)$, is almost 20 times higher than the corresponding number for interval I_4, $N(I_4)$, to compensate for the ratio between 4.8% and 95.2%.

C. Quantitative Analysis for Depth 3

The quantitative analysis for depth 3 is more intricate. So, we have to explain specific variations of the increments, as indicated above in the context of complete tree generation. These variations and the interpretation of the results for depth 3 focus on specific situations with characteristic properties, in order to investigate the respective capabilities of the competing backup rules.

1) Bias in Favor of the First Player: In addition, there is another issue to consider when expanding depth 2 searches to depth 3 according to our tree model. Unlike at even depths, searches to odd depths involve the application of value increments by the side to move first once more than by the other side. As a consequence, the distribution of values over the leaf nodes in a search to an odd depth has a strong bias in favor of the side to move first. This effect is particularly pronounced when the leaf node values are built by adding two increments in one and only one increment in the other direction. In order to compensate for this bias, we adapted the value of the root node, which is the starting point for the value assignment procedure, so that the set of values assigned to the leaf nodes is roughly balanced between those favoring one or the other side. For a maximum increment of 3, the average increase of the root node value at the leaf nodes is 1.5, the average of all increments. Thus, we computed two depth 3 searches according to the tree model with maximum increment 3, one starting with a root value of -1, the other with a value of -2. The result was that 0.5% of all critical cases favor Product for the root value of -2, while 0.8% of all critical cases favor Minimax for the root value of -1. Hence, there is some evidence from these searches that Minimax is competitive to Product under these conditions.

2) Degree of Advantage of One Player—Variations of the Root Node Value: We conjectured that the respective quality of the two backup rules under investigation could depend on the degree of advantage of one player over the other in a given position. This is important from a strategic perspective for games where the ultimate result is either a win or a loss, or possibly, a draw, such as chess. If one side has a big advantage, it should tend to avoid any risk and counter-play. The other side should generally look for complications, seeking possible counter-chances, if any. Our examinations should show, which of the competing backup rules is more suitable to follow a risk-avoidance or a chance-seeking strategy, respectively.

In our models, variations of the root node value may lead to situations where one of the players is favored substantially by most of the leaf node positions, or the positions may be more or less evenly distributed, or they may be somewhere in between these two pronounced situations. So, the value assigned to the root node has been varied, which determines the distribution of values at the leaf nodes according to the assumptions underlying the tree model. Depending on this value, the proportion of positive to negative leaf node scores changes.

3) "Quiet" Versus "Nonquiet" Situations—Variations of the Maximum Increment Value: Another category of situations is important from the perspective of the impact that individual move decisions may have. In "quiet" positions,[13] winning may require the accumulation of several minor advantages; in other

[13]This is conceptually related to the notions *quiescent* and *quiescence search* in the context of computer chess as coined already by Turing [25].

TABLE II
PERCENTAGE DIFFERENCES FAVORING MINIMAX (POSITIVE NUMBERS) OR PRODUCT (NEGATIVE NUMBERS). ACCORDING TO VARYING VALUE DISTRIBUTIONS AND ROOT NODE VALUES (-4 TO 2)

Max. Increments	Root Center	Root -4	Root -3	Root -2	Root -1	Root $+1$	Root $+2$	Mean
$(4, 2)$	-1	-3.9%	-4.1%	-3.1%	-1.5%	-3.1%	-0.8%	-2.75%
$(3, 3)$	-1.5	-4.1%	-3.1%	-0.5%	$+0.8\%$	$+0.0\%$	-1.0%	-1.32%
$(2, 4)$	-2	-3.6%	$+0.4\%$	$+3.1\%$	$+1.0\%$	-0.8%	-1.2%	-0.83%
$(1, 6)$	-3	$+3.9\%$	$+4.0\%$	$+1.7\%$	-0.3%	-1.1%	-1.3%	$+1.12\%$
$(1, 5)$	-2.5	$+1.2\%$	$+4.1\%$	$+2.5\%$	$+0.0\%$	-1.0%	-1.1%	$+0.95\%$
$(1, 4)$	-2	-3.7%	$+2.3\%$	$+3.3\%$	$+0.5\%$	-0.8%	-1.1%	$+0.08\%$
$(1, 3)$	-1.5	-3.4%	-3.4%	$+3.0\%$	$+1.3\%$	-1.0%	-1.0%	-0.75%

words, a suboptimal decision may not immediately lead to a very bad situation. In "nonquiet" positions, in contrast, there may exist an increasing number of situations with a forced move, where a player must make a specific move, since otherwise his position will become much worse. Also for this category of situations our examinations should show which of the competing backup rules is more suitable, if any.

The maximum value increment, which regulates the score assignment to nodes on deeper levels from their predecessors in the tree, has been strongly reduced to a value of 3 from a value of 8 in the original tree model. Moreover, this reduction has been done identically for all three search levels. Since this increment is comparably small, it appears worthwhile to examine also some larger increments. An increase of the maximum increment at some levels in the search tree would have to go together with a decrease at some other levels. In order to produce most distinct distributions of values, we varied the maximum increment on the last (third) level of the search tree, compensated by a joint change of the maximum increment (in reverse direction) on the two other levels of the search tree.

These variations lead to varying score differences between adjacent leaf nodes, depending on the maximum increment value at the last level of the search tree. In case of a comparably low maximum increment value, the overall situation tends to appear "quiet" in the sense that only small advantages can be obtained, and there are small differences of the values between competing candidate moves. In case of a comparably high maximum increment value, the overall situation tends to appear rather "nonquiet" in the sense that occasionally larger advantages can be obtained, and also the differences between the values of competing candidate moves can be larger.

D. Quantitative Results for Depth 3

Our quantitative results of comparing the two backup rules for depth 3 are based on exhaustive analyses for several combinations of these parameters. So, they focus on the two categories of situations as described above, and their combined occurrences.

In Table II, varying value distributions and root node values are given, together with the corresponding results of searches of the competing backup rules, indicating the preferable backup rule in terms of percentage differences in critical positions. The first two columns contain specifications of the value distributions for building the search tree. In the first column, the pairs of maximum increment parameters are listed, first the one limiting increments for depths 1 and 2, then the one limiting the increment for depth 3. In the second column, the approximate value for the root node is given,[14] for which the value distribution resulting from these increment parameters is mostly balanced between the players. For example, (4,2) in the first line below the header in the first column, indicates a maximum increment of 4 at depths 1 and 2, and a maximum increment of 2 at depth 3. For this combination of increments, the (approximate) value for the 'most central' root node amounts to -1, as given in the first line below the header in the second column. The remaining columns in this table, except the header line, contain percent values like the one given above for the depth 2 result. They express preferences in favor of Minimax (positive numbers) or Product (negative numbers), respectively, for the combination of root value (-4 to $+2$, according to the header line) and search tree level increments (first and second column) corresponding to their position in the table. The last column lists mean values of the percent values given to the left in each line, i.e., it averages over the values in columns 3 to 8 in the same line.

As shown in the header line of the table, we varied the scores of the root node between -4 and $+2$, and we picked a small set of pairs of maximum increment values for depth 3 and for the two preceding levels, respectively. We chose seven combinations, which exemplify the kind of constellations already referred to in the comparison between depth 2 and depth 3 searches above (see the leftmost column in Table II). They comprise "minimal" changes, that is, incrementing at all three levels in a roughly comparable manner up to a maximum value between 2 and 4 (see the first three lines below the header in the table), and 'maximum diversion' by allowing the increment at levels 1 and 2 to be at most 1, while varying the maximum increment at level 3 between 3 and 6 (the other four lines).

The numbers in Table II provide evidence for pronounced differences:

- The relative advantages of the competing backup rules vary depending on the value distribution of the leaf nodes, which is determined through the given parameters. In each line of this table, the root values are varied. In the first line below the header, the one with the combinations of maximum values of increments—(4,2)—all values are in favor of Product. In all the other lines below, the

[14]It is an approximation for which the values of the resulting leaf nodes are mostly balanced between positive and negative ones for this combination of increments.

number of cases favoring Minimax is almost the same as those favoring Product. The respective advantage depends on the varying root values.
- The table also shows that the differences between the competing backup rules can become relatively large—more than eight percentage points between the two most extreme cases. There are a few cases in favor of each of the competing backup rules, which indicate a preference in the order of four percentage points (e.g., the cell in the second line below the header and the third column indicates a preference for Product by 4.1%, while the cell in the fifth line below the header and the fourth column expresses a preference for Minimax by the same amount).

Concerning the relative advantages of the competing backup rules across varying situations, the data listed in Table II indicate the following preferences.
- In comparably "quiet" situations (see the first line below the header in the table), Product is superior.
- In "nonquiet" situations, Minimax is superior, with increasingly larger sets of values at the leaf nodes. This can easily be seen in the rightmost column in Table II; the higher the increment at level 3, the more pronounced becomes the advantage of Minimax.
- Independently of the proportion of "quiet" versus "nonquiet" situations, Table II illustrates that Product is increasingly better than Minimax, the more the root values deviate from the values in column 2. That is, if one player has a clear winning position, Product becomes slightly better. This effect can be observed in the table by comparing percentage points in favor of Product in each line: starting from relatively low values at those places in the middle (root values close to the value in the same line in column 2), percentage points increasingly favor Product for cells that are located more to the left or to the right of the 'central' value.
- The results obtained for the trees with the most even distribution of positive and negative values, that is, for root values close to the center of the leaf node value distribution (according to the values in column 2), favor Minimax. In particular, the percentage value in each line is highest if root value and root center are identical, e.g., -1.5% in the first column below the header for root value and root center both equal to -1.

Altogether, these results demonstrate that each of the two competing backup rules has systematic advantages and disadvantages in comparison to the other, depending on properties of the leaf node value distributions in the search tree.

VI. Simulation Study

Unfortunately, the systematic analyses above cannot be extended further due to a combinatorial explosion. Therefore, we present also a simulation study that allows less shallow searches as well. In order to make this study as general as possible, synthetic search trees are constructed that fulfil the properties of our game tree model (instead of defining any simplified specific game).

A. Tree Generation for Simulated Games

Based on the general tree generation approach explained above in Section III-C, let us elaborate on the specifics of the tree generation for simulated games, in contrast to the complete tree generation explained in Section V-A. Since no complete enumeration of all trees is to be done, only the general recursive tree construction is relevant, but increment values according to (3) have to be assigned instead. Instead of the enumeration, it requires concrete WIN and $LOSS$ values to be assigned consistently with (1)—we used $c = 16$ in the course of this tree generation. So, no propagation of probabilities for all relevant cases is required.

Both the assignment of increments and of WIN and $LOSS$ values are stochastic events in the course of the simulations. Generally, every stochastic event in the tree is simulated by a call to a *pseudorandom number generator*, parameterized independently of the relative frequencies achieved earlier in the tree generation process.

B. Design of the Simulation Study

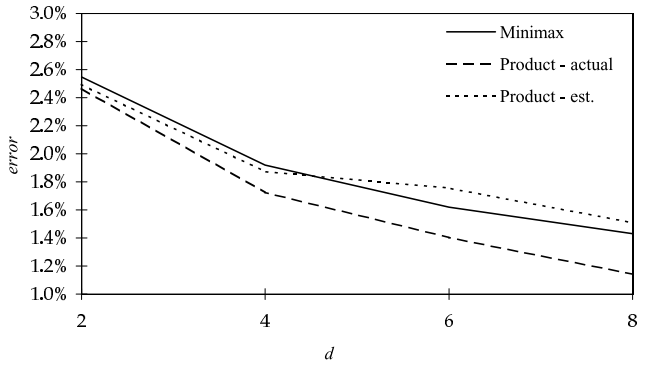
In order to compare Minimax with Product, we performed a simulation study based upon previous ones described in [9]. However, the simulation study as reported below deals with simulated game contests.

Generally, synthetic game trees satisfying the assumptions of the model in Section III were generated according to the method described in Section VI-A with the specifics elaborated on above. Then searches of different depths were performed in these game trees. These searches computed minimax values $MM^d_f(n)$ and values $PP^d_{f'}(n)$ according to product propagation, respectively, for all the possible depths d in such a tree.[15]

This simulation study had two distinct parts, depending on how the values from f were mapped for their use by Product. In the first part, we mapped using $f'(n) = w_1(f(n))$, i.e., $c = 1$. That is, f' as used by Product in this part of the simulation study provided *estimates* of the probabilities to win, which is realistic in practice. A special feature of our synthetic model is that probabilities to win are actually known. In the following, we refer to these as *actual* probabilities. In order to see how well Product can perform in the limit, we provided it with these probabilities in the second part of our simulation study. More precisely, for the given value of $c = 16$, which parameterized the model, we mapped through $f' = w_{16}(f(n))$ into the probabilities as given in this model. Note, that this can normally not be done in practical games like chess, where the probabilities may just be estimated.

The goal of performing these searches was to gather data for the errors of move decisions and the scores of game contests for

[15]The static evaluation function f is characterized in such a simulated tree search by the corresponding parameters. *Backward pruning* was used to avoid unnecessary effort for the minimax searches; see for instance [26] for a description of the alpha-beta algorithm. This pruning complicates the reproducibility of the trees, so that we stored them in memory. Within such a stored tree of depth d_g, all searches possible were performed, where the maximum search depth is d_g, of course, and more searches to depths smaller than d_g are possible. Another approach to random tree generation during the search is described in [27, Sec. IV].



$d_g = 20, b = 2, h \in \{-42, \ldots, -1, +1, \ldots, +42\}, a = 9, c = 16, e \approx 0.10$

Fig. 6. Quality of move decisions by Minimax, Product-actual using actual probabilities, and Product-est using estimated probabilities.

the comparison of Minimax with Product. So, let us define such game contests within the synthetic trees. Both Minimax and Product perform full-width searches to the *same* given depth. We had both Minimax and Product search to the same depths, although this is unfair to Minimax. Each game was played twice to allow both Minimax and Product to move first. In our synthetic trees, the nodes do not represent real game positions, but they are just characterized through a true value and a corresponding heuristic estimate, which are related through the function e_c as defined in (1).

The model underlying our synthetic trees does not contain any real terminal nodes (like check-mate in chess). Therefore, we had to define another means of terminating a game. In fact, few of the games in real tournament chess end with actual draw or check-mate, since either one player resigns or both players agree upon a draw, or the games are adjudicated. We chose the last of these possibilities for our synthetic games. In fact, adjudication is guaranteed here to be perfect, since the true values of all "positions" are known.

C. Results

First, we present the results from our simulation study on the respective quality of move decisions and then the results from game contests.

1) Quality of Move Decisions: Fig. 6 shows the quality of move decisions for increasing search depths that occurred in the game contests played according to our simulation study design.[16] For a given depth d, the frequency of making a wrong move decision is given for Minimax, Product-actual with the actual probabilities, and Product-est using the estimated probabilities. The errors made by Minimax are always slightly more frequent than those of Product with the actual probabilities.[17] Compared with Product using the estimated probabilities, Minimax makes a wrong move decision slightly more frequently for lower search depths, but less frequently for higher search depths. Note, however, that the differences among all three competitors are rather small.

So, it is difficult to tell from these data, which backup rule propagates the given values "better" in the sense of making better decisions based on them. Product with the actual probabilities makes move decision errors less frequently, but the probabilities can only be estimated in practice. With estimated probabilities, however, the case is undecided for searching to the same depth. Much as in the results of the systematic analysis above, Product is better for depth 2, but with increasingly deeper search, Minimax tends to become better.

In reality, Minimax can search much deeper with the same amount of nodes due to its well-known pruning algorithms. How much deeper Minimax can search, depends primarily on the successor ordering. For random ordering, the search depth can roughly be increased by a factor of 4/3 [4], for very strong ordering as observed in computer chess practice, the search depth can nearly be doubled (it approaches the optimal factor of 2) [17]. For example, the move decision error of Minimax in Fig. 6 at depth 6 (which is certainly achievable with simple move ordering heuristics) is lower than that of Product-est at depth 4. The decision error at depth 8 (when pruning might allow the search depth to be doubled, which is not completely unrealistic) is even clearly lower than the error of Product-actual, which is not available in practice. So, under practical conditions, the move decisions of Minimax searching deeper than Product are clearly better than those of Product, even when actual probabilities were available to Product.

Pearl [5] suspects that if judged against absolute standards of WIN-$LOSS$ status, a less pronounced improvement with increasing search depth may be exhibited by minimaxing than observed in computer chess practice. In contrast to typical chess positions, in our simulations the true value of each position is exactly known, but still this pronounced improvement is shown (see Fig. 6).

2) Results from a Game Contest Between Minimax and Product: We had Minimax play a game contest against Product in synthetic trees according to the model of Section III. This simulation study has two parts, one where Product uses estimated probabilities, while in the other the actual probabilities to win are available to Product.

In the first part of our simulation study, we had Minimax play a game contest against Product where it had only estimates of the probabilities available. Table III shows the results under this realistic condition. Still, Product was allowed searching to the same depth as Minimax.

While Minimax lost against Product for search depth 2, it won for depths 3, 5 and higher. For instance, when both Minimax and Product played to the same search depth $d = 2$, Minimax won only 4,834 from a total of 393,213 game pairs, while Product won 5,212. The results for depths 2 and 3 are consistent with our analyses above.

Much as in the results of the game contest reported by Nau [14], however, the overall score is only marginally better. In this example, it amounts to 49.95% of all the game pairs. Therefore, Table III also contains the scores of the *critical* games, where one player was able to win an initial position for both sides. For these, the advantage shows up more clearly (much as in [15]);

[16]The various parameters are explained in Section III. e is the average probability of error of static evaluation in these simulation runs.

[17]In fact, when the actual probabilities are known, product propagation is a better backup rule than minimaxing. In contrast, minimaxing is favored by low errors [12].

TABLE III
RESULTS OF A GAME CONTEST BETWEEN PLAYERS MINIMAX AND PRODUCT—WITH *Estimated* PROBABILITIES—IN SYNTHETICALLY GENERATED GAME TREES, BOTH MINIMAX AND PRODUCT SEARCHING TO THE SAME DEPTH d

d	Total Game Pairs	Critical Game Pairs	Wins by Minimax	Wins by Product	Minimax % Wins Total	Minimax % Wins Critical	Significance
2	393,213	10,046	4,834	5,212	49.95%	48.12%	0.00008
3	196,605	7,820	4,027	3,793	50.06%	51.50%	0.00407
4	98,301	3,326	1,663	1,663	50.00%	50.00%	0.50000
5	49,149	2,038	1,164	874	50.30%	57.11%	0.00000
6	24,573	833	512	321	50.39%	61.46%	0.00000
7	12,285	516	292	224	50.28%	56.59%	0.00138
8	6,141	241	139	102	50.30%	57.68%	0.00858

TABLE IV
RESULTS OF A GAME CONTEST BETWEEN PLAYERS MINIMAX AND PRODUCT—WITH THE *Actual* PROBABILITIES—IN 11PSYNTHETICALLY GENERATED GAME TREES, BOTH MINIMAX AND PRODUCT SEARCHING TO THE SAME DEPTH d

d	Total Game Pairs	Critical Game Pairs	Wins by Minimax	Wins by Product	Minimax % Wins Total	Minimax % Wins Critical	Significance
2	393,213	9,395	4,284	5,111	49.89%	45.60%	0.00000
3	196,605	6,253	2,812	3,441	49.84%	44.97%	0.00000
4	98,301	2,681	1,053	1,628	49.71%	39.28%	0.00000
5	49,149	1,594	669	925	49.74%	41.97%	0.00000
6	24,573	585	218	367	49.70%	37.26%	0.00000
7	12,285	372	127	245	49.52%	34.14%	0.00000
8	6,141	148	51	97	49.63%	34.46%	0.00000

e.g., for the depth 2 part of the contest, Minimax won 48.12% of the critical game pairs.

In order to see whether these results are significant, we used the same approach for statistical analysis as [15]. We considered the null hypothesis that the number of pairs of wins is a random event with probability one half, i.e., that each method is equally good. The *sign test* is appropriate here, especially since it requires no special assumptions about distributions of the samples. The Significance column in Table III (and also in Table IV) gives the probability that the data is consistent with the null hypothesis. Small numbers (below, say, 0.05), indicate that the deviation away from 50% in the percentage of wins is unlikely to be from chance fluctuations. Large numbers indicate that from this data one cannot reliably conclude which method is best.[18]

For the critical games only, the results are statistically significant, e.g., the probability that the result is due to chance fluctuation for $d = 2$ is smaller than 0.01%. In terms of the total number of game pairs, however, the results are not statistically significant.

In the second part of our experiment, we had Minimax play a game contest against Product where it had the actual probabilities available. Table IV shows the results for those trees in which also the results of Fig. 6 were gained. Under this unrealistically favorable condition and being allowed to search to the same depth as Minimax, Product defeated Minimax. For the critical games only, the results are clearly statistically significant. For all depths, the probability that the result is due to chance fluctuation is smaller than 0.005%.

In summary, even when searching to the same depth (which is very unfair to Minimax because it could search far deeper due to backward pruning), Product performed mostly worse under the more realistic condition that only estimated probabilities are available. With increasing search depths starting at depth 5, the superiority of Minimax over Product seems to become more and more pronounced. Even when the actual probabilities to win were available to Product, its performance was not at all good enough to envisage that it could be better than Minimax using one of its pruning algorithms.

[18]0.00000 in these columns indicates that significance is even smaller than this number, i.e., a highly significant result.

VII. CONCLUSION

This paper shows through systematic analyses and in a simulation study using synthetic game trees (modeled according to properties found in chess game trees) that product propagation can be better than minimaxing only under specific conditions:

- the searches are shallow, or;
- the actual probabilities to win are available.

Apart from the second constellation, which constitutes a rather specific case not present in practical games, these results show complementary capabilities of the competing backup rules, which are unnoticed in the literature so far. While product propagation leads to the better decision quality, minimaxing is the superior backup rule.

Product tends to make better decisions more frequently, under realistic assumptions modeled after real game-playing programs. As a consequence from this theoretical investigation, the decision quality based on product propagation may still make it a viable alternative to minimaxing for game trees where only very shallow searches are affordable. When there is a huge number of alternatives to consider and only a single application of the competing backup rules in each branch can be made, the decision quality based on product propagation is better than that based on minimaxing. This conclusion from our own study is also supported by the result in [10] that minimaxing works better with smaller than with larger branching factors.

However, Minimax can usually search deeper than Product due to the availability of effective pruning algorithms. Even

under the conditions where product propagation provides the better backup values, minimaxing can come up with even better values from its deeper searches.

In addition, we found new evidence that minimaxing profits from better quality of the evaluation function used. However, if one player has a clear winning position, Product becomes slightly better. With respect to "quiet" versus "nonquiet" positions, i.e., situations with smaller versus larger differences between heuristic values of the leaf nodes, Product or Minimax tend to be slightly better, respectively.

Finally, the previous view that minimaxing—in contrast to product propagation—should be effective specifically against fallible opponents, is questionable. We have illustrated that product propagation sometimes leads to errors in the "hope" for blunders by fallible opponents, which top players rarely make. We also showed in our game tree model with known true values of positions a pronounced improvement with increasing search depth of minimaxing as judged against these absolute standards, in contrast to a previous conjecture in [5]. So, we could not find evidence that product propagation should be a better backup rule than minimaxing in current game-playing practice with deep searches.

REFERENCES

[1] J. Schaeffer, *et al.*et al., "Checkers is solved," *Science*, vol. 14, pp. 1518–1522, Sep. 2007.

[2] H. Kaindl, "Searching to variable depth in computer chess," in *Proc. Int. Joint Conf. Artif. Intell. 1983*, Karlsruhe, Aug. 1983, pp. 760–762.

[3] O. David-Tabibi and N. S. Netanyahu, "Extended null-move reductions," *CG 2008*, ser. LNCS 5131, New York: Springer-Verlag, 2008, pp. 205–216.

[4] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Reading, MA: Addison-Wesley, 1984.

[5] J. Pearl, "On the nature of pathology in game searching," *Artif. Intell.*, vol. 20, no. 4, pp. 427–453, 1983.

[6] D. Nau, "Pathology on game trees: A summary of results," in *Proc. 1st Nat. Conf. Artificial Intelligence*, Stanford, CA: 1980, pp. 102–104.

[7] D. Beal, "An analysis of minimax," *Advances in Computer Chess 2*, M. Clarke, Eds., Edinburgh, U.K.: Edinburgh Univ. Press, 1980, pp. 103–109.

[8] B. Wilson, I. Zuckerman, A. Parker, and D. S. Nau, "Improving local decisions in adversarial search," in *Proc. ECAI'12*, 2012, pp. 840–845.

[9] A. Scheucher and H. Kaindl, "Benefits of using multivalued functions for minimaxing," *Artif. Intell.*, vol. 99, no. 2, pp. 187–208, 1998.

[10] D. S. Nau, M. Lustrek, A. Parker, I. Bratko, and M. Gams, "When is it better not to look ahead?," *Artif. Intell.*, vol. 174, pp. 1323–1338, 2010.

[11] J. Slagle and P. Bursky, "Experiments with a multipurpose, theorem-proving heuristic program," *J. ACM*, vol. 15, no. 1, pp. 85–99, 1968.

[12] P. Chi and D. Nau, "Comparison of minimax and product backup rules in a variety of games," *Search in Artif. Intell.*, L. Kanal, V. Kumar, Eds., New York: Springer-Verlag, 1988, pp. 450–471.

[13] D. Nau, "On game graph structure and its influence on pathology," *Int. J. Comput. Inf. Sci.*, vol. 12, no. 6, pp. 367–383, 1983.

[14] D. Nau, "Pathology on game trees revisited, and an alternative to minimaxing," *Artif. Intell.*, vol. 21, 1–2, pp. 221–244, 1983.

[15] D. Nau, P. Purdom, and H. Tzeng, "Experiments on alternatives to minimax," *Int. J. Parallel Programm.*, vol. 15, no. 2, pp. 163–183, 1986.

[16] H. Horacek and H. Kaindl, "An analysis of decision quality of minimaxing vs. product propagation," in *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics*, San Antonio, TX, USA, 2009, pp. 2642–2648.

[17] H. Kaindl, R. Shams, and H. Horacek, "Minimax search algorithms with and without aspiration windows," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 12, pp. 1225–1235, Dec. 1991.

[18] E. Baum, "On optimal game tree propagation for imperfect players," *Proc. 10th Nat. Conf. Artif. Intell.*, San Jose, CA, USA, 1992, pp. 507–512.

[19] D. Stern, R. Herbrich, and T. Graepel, "Learning to solve game trees," in *Proc. 24th Int. Conf. Mach. Learn.*, New York, NY, USA, 2007, pp. 839–846, http://doi.acm.org/10.1145/1273496.1273602

[20] A. Saffidine and T. Cazenave, "Developments on product propagation," in *Proc. CG 2013*, LNCS 8427, New York: Springer-Verlag, 2013, pp. 100–109.

[21] M. H. Winands, J. W. Uiterwijk, and H. J. van den Herik, "PDS-PN: A new proof-number search algorithm–application to lines of action," in *Proc. CG 2002*, LNCS 2883, New York: Springer-Verlag, 2003, pp. 61–74.

[22] H. Baier and P. D. Drake, "The power of forgetting: Improving the last-good-reply policy in monte carlo Go," *IEEE Trans. Comput. Intell. AI Games*, vol. 2, no. 4, pp. 303–309, 2010.

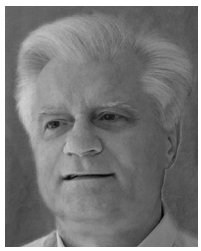
[23] H. Baier and M. H. M. Winands, "Monte-carlo tree search and minimax hybrids," in *Proc. IEEE Conf. Comput. Intell. Games*, Niagara Falls, ON, Canada, 2013, pp. 129–136.

[24] H. Horacek, "Reasoning with uncertainty in computer chess," *Artif. Intell.*, vol. 43, pp. 37–56, 1990.

[25] A. Turing *et al.*, "Digital computers applied to games," *Faster than Thought*, B. Bowden, Eds.London, U.K.: Pitman, 1953, pp. 286–295.

[26] D. Knuth and R. Moore, "An analysis of alpha-beta pruning," *Artif. Intell.*, vol. 6, no. 4, pp. 293–326, 1975.

[27] R. Korf and D. Chickering, "Best-first minimax search," *Artif. Intell.*, vol. 84, 1–2, pp. 299–337, 1996.



Hermann Kaindl (SM'99) received the Dipl.-Ing. degree in Informatik (computer science) in 1979, and the doctoral degree in 1981, both from the Technical University of Vienna (now named TU Wien), Vienna, Austria.

Currently, he is the Director of the Institute of Computer Technology at TU Wien, which he joined in early 2003 as a Full Professor. Prior to moving to academia, he was a Senior Consultant with the division of program and systems engineering, Siemens AG, Austria. There he has gained more than 24 years of industrial experience in software development. He has published five books and more than 200 refereed papers in journals, books, and conference proceedings.

Dr. Kaindl is a member of the AAAI, a Distinguished Scientist member of the Association for Computing Machinery, and a member and Executive Board member of the Austrian Society for Artificial Intelligence.

Helmut Horacek received the Dipl.-Ing. degree in Informatik (computer science) in 1979, and the doctoral degree in 1982, both from the Technical University of Vienna, Vienna, Austria. He also holds a Venia Legendi for artificial intelligence at Saarland University, Saarbrücken, Germany.

He is a Researcher at the German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany, and a Lecturer with the Department of General Linguistics, Saarland University. He has worked on research projects in several aspects of natural language processing at the universities of Vienna, Hamburg, Bielefeld, and Saarland University. His research interests include natural language generation, explanation, computational models of natural arguments, models of discourse, intelligent tutoring systems, and search methods.

Anton Scheucher received the Dipl.-Ing. degree in Informatik (computer science) from the Technical University of Vienna, Vienna, Austria, in 1989, the M.S. degree in computer science from Michigan State University, Ann Arbor, in 1990, and the MBA degree in business administration from the University of Economics and Business Administration of Vienna in 1992.

Between 1992 and 2004, he was with Accenture, where he was responsible for the management and consulting of large projects with project teams between 10 and 100 employees in the banking industry in Austria and Germany. Since 2004, he has been with Raiffeisen Bank International, Vienna, where he is currently the Program Manager responsible for the FATCA Group Program with impact on all units of the Raiffeisen Bank International Group in 16 countries. In addition, he is a certified "Project Management Professional" (PMP) and a certified "ScrumMaster".