ELSEVIER

Artificial Intelligence 99 (1998) 187–208

Artificial Intelligence

# Benefits of using multivalued functions for minimaxing

Anton Scheucher [a], Hermann Kaindl [b,*]

[a] *Bergstraße 24, A-8600 Bruck/Mur, Austria*

[b] *Siemens AG Österreich, PSE KB1 Ref3, Geusaugasse 17, A-1030 Wien, Austria*

Received August 1993; revised July 1997

**Abstract**

Minimaxing has been very successful in game-playing practice, although a complete explanation of why it has been that successful has not yet been given. In particular, it has not been shown why it should be useful—as it is in practice—to use *multivalued* evaluation functions. Such functions have many distinct values as their result and can discriminate between positions according to the heuristic knowledge represented in these values. In this paper, we modify a basic *pathological* model by adding two assumptions regarding multivalued evaluation functions. These assumptions, *non-uniformity of error distribution* and *dependency of heuristic values*, directly relate to the properties of multivalued evaluation functions as used in practice. Simulation studies of our multivalued model have exhibited sharp error reductions for deeper searches using minimaxing. This behavior corresponds to observations in practice. The error reductions are primarily due to the *improved evaluation quality* as search depth increases. This phenomenon of lower probability of static evaluation errors with increasing search depth is revealed through our model, 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 which can be more reliably evaluated by a multivalued function with the assumed properties. 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. © 1998 Elsevier Science B.V.

*Keywords:* Game trees; Minimaxing; Multivalued evaluation functions; Improved evaluation quality; Simulation studies

* Corresponding author. Email: hermann.kaindl@siemens.at.



*PII* S0004-3702(97)00073-8

## 1. Introduction

Commercial chess programs and computers are now widely available that defeat most of their human opponents. Recently, the special chess machine Deep Blue even defeated the highest-rated human chess player for the first time in a match consisting of several games under tournament conditions. Since 1994, the checkers program Chinook is even the official man–machine world champion [31]. However, it is not completely known *why* these machines play that well—due to deep searches using *minimaxing*. After all, some theoretical studies have even shown to the contrary that minimaxing can result in *pathological* behavior, i.e., the deeper the minimax search the worse the result. Thus, there has not yet been a full explanation of the dramatic benefits of using minimaxing for deeper and deeper searches in game-playing programs as observed, e.g., in computer chess and checkers practice.

We think that this lack of insight is because until now no one has seriously investigated the role of using *multivalued* functions for heuristic evaluation and back-up, i.e., functions with more heuristic values than "true" values in the sense of *WIN* or *LOSS*. While such functions have *many distinct values* as their result, most theoretical studies have restricted the investigated models to only *two* heuristic values. Other studies that have investigated multivalued functions have not assigned any special meaning to different values. Therefore, no difference has been found in the behavior compared to the corresponding model with two values.

It seems clear, however, that some important aspects observed in practice must have been lost in these models. In fact, we are unaware of any single chess, checkers, or kalah program that is confined to just two heuristic values (or three, including draws). For instance, how should such a program discriminate between a position with a slight advantage from one that is clearly won? Therefore, we performed an experiment using an adapted tournament chess program confined to three heuristic values (see the Appendix). The results of this experiment show that the benefits of searches using minimaxing are much more pronounced with multiple heuristic values.

This is the first work showing the reason why the use of a *multivalued* evaluation function is important for minimaxing. Such functions are ubiquitous in practical applications of minimaxing, and our model as introduced in this paper captures what is essential in using them in order to make minimaxing strongly beneficial. Both the assumptions of our model and the results from its application correspond quite closely to what is observed in practice.

This paper is organized in the following manner. First, we sketch the background of this work in order to make this paper self-contained. Then we describe our multivalued model. It contains two important assumptions that are motivated and described in detail. After that, we elaborate on the experiment design for examining the behavior of this model. Based on this experiment design, we present the results of simulation studies that investigate the behavior of the static and dynamic evaluation errors as well as the errors of move decisions with increasing search depth. Moreover, we show the influence of different granularity in the sense of the number of distinct values. Finally, we compare our approach with related work in the literature.

## 2. Background

Except in very rare cases, there is no practical way of determining the exact status (the *true* value) of non-goal nodes. Therefore, it is usually necessary to resort to heuristic estimates. These can simply be *point values* that range over an interval of integer numbers (or, for theoretical purposes, sometimes real numbers). Such *heuristic* 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 $n$ with some error.

Given such an evaluator, the question remains how its values can be used for making reasonable decisions. When exactly one ply is searched, it seems clear that the choice is one of possibly several actions leading to the "best" value (maximum or minimum). But since the immediate application of $f$ to the children of the given node usually does not lead to good decisions, 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 and, how the heuristic values should be backed up (i.e., propagated) towards the given node's children. In *two-person games* with *perfect information* the most successful approach for the back-up of values has been *minimaxing*, although there has been some theoretical doubt on its usefulness. In the following, we assume that $f(n)$ assigns a value to a node $n$ from the viewpoint of the moving side at $n$.

**Definition 1.** A *minimax value* $MM_f(n)$ of a node $n$ can be computed recursively as follows (in the *negamax* formulation):

(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 $n$.

Since the *depth* of such searches will be important for the purpose of this paper, we also define a special case of using this rule.

**Definition 2.** $MM_f^d(n)$ is the minimax value of node $n$ resulting from exactly $d$ applications of recursion (2) in Definition 1 in every branch of the search tree.

In fact, $MM_f^d(n)$ is the minimax value from a *full-width search* of the subtree rooted at $n$ to a uniform depth $d$.

## 3. A multivalued model

For our multivalued model we assume that

(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,[1] and
(4) heuristic values $h$ (assigned to a node $n$ by a static evaluation function $f(n)$) are elements of the set $\{-h_{\max}, \ldots, -1, +1, \ldots, +h_{\max}\}$.[2]

These assumptions are derived from Pearl's [29] basic *pathological* model.

At all different depths we use the *same* evaluation function, and its general error is *independent* of the depth. In Pearl's model of "improved visibility" [29], the accuracy of the evaluation function has to improve by at least 50 percent per move cycle in order to avoid pathological behavior. In chess, however, no such improvement of accuracy can be seen when considering the static evaluation functions of actual programs. Computer chess experts agree that the endgame evaluators are in general worse than those for the midgame, but this very slight increase of error is difficult to quantify. In fact, within the horizon of one search the quality of an evaluator remains nearly constant.

In addition to these basic assumptions, we specify in the next two sections some *very important* characteristics of an evaluation function as observed in strong chess and checkers programs. The first characteristic—*non-uniformity of error distribution*—is responsible that the different values of such a multivalued function have different meaning. The second characteristic—*dependency of heuristic values*—is utilized in practice for efficient incremental computation of such a heuristic evaluation function.

### *3.1. Non-uniformity of error distribution*

First, we discuss the role of multivalued evaluation functions in practice and how it can be modeled using probabilities. More precisely, we define "probabilities to win/lose" and then error probabilities. We argue that the errors of the heuristic values from such a static evaluation function are not uniformly distributed in practice, so that we model them here through a non-uniform error distribution that is based on the distribution of the "probabilities to win/lose".

In practice, a multivalued evaluation function is used to express the degree of "goodness" of a game situation in chess, checkers, or kalah for the moving player and to include domain-specific knowledge that measures factors such as material balance, mobility, etc. The values of these components are typically combined by a weighted sum, although non-linear combinations are sometimes also used. The resulting values induce a partial order on the various positions of a game according to their worth in the sense of "goodness" or "strength" for one side. High estimates (typically positive values) indicate a "strong" position, whereas low estimates (typically negative values) are used for "weak" positions.

---

[1] A nonterminal node is won if at least one of its child nodes is won.

[2] 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 $h_{\max}$. Details regarding the calculation of this value can be found in [32]. Evaluation functions used in successful programs have actually finer-grained evaluation functions since they additionally evaluate positional aspects. Using such additional knowledge allows to discriminate better between positions of different "goodness", which will be discussed in more depth below. 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 (or checkers). So, for showing the major effects in this paper we use $h_{\max} = 42$.

For modeling purposes, it is important to precisely define the meaning of these intuitive notions. While the real meaning of the point values is not completely clear, probabilities can be used for modeling purposes. For instance, heuristic point values have sometimes been related to the notion of "probability to win", i.e., the probability of winning the game from positions having a specific heuristic value [30].[3] 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. Pearl [30, p.360] proposed such a function based on the arc tangent function in order to be able to use *product propagation* with the common estimators. In a more practical approach, Horacek [7] informally used a function with similar shape in order to handle reasoning with uncertainty in a computer chess program.

In our model, we want to have a more general function that relates values to probabilities. Pearl's function based on the arc tangent cannot be parameterized, so we generalize it to a set of "probability to win" distributions $w_c(h)$ by introducing a parameter $c$.[4] The parameter $c$ allows us to specify the slope of the curve. For simplicity, we assume a continuous range of heuristic values at this point.

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

In the experiments, we use the discrete set $\{-h_{\max}, \ldots, -1, +1, \ldots, +h_{\max}\}$ as given in the basic assumptions above. Analogously, a "probability to lose" can be defined by $1 - w_c(h)$.

In the following, we are mainly interested in the *errors* made by such an evaluation function. Errors occur when a node with true value *WIN* (*LOSS*) is assigned a negative (positive) heuristic value. In order to define such errors more precisely, we need an auxiliary definition first for mapping heuristic values to true values.

**Definition 3.** The utility function $u(h)$ maps heuristic values to true values in the following way:

$$u(h) = \begin{cases} WIN, & h \in \{1, \ldots, h_{\max}\}, \\ LOSS, & h \in \{-h_{\max}, \ldots, -1\}. \end{cases}$$

Based on this definition, we characterize the error made by an evaluation function $f$ as a probability of error.

[3] A general discussion of different meanings of "probability to win" can be found in [8].

[4] $\int_{-\infty}^{\infty} w_c(h)\, dh = 1$, see [32] for a proof.

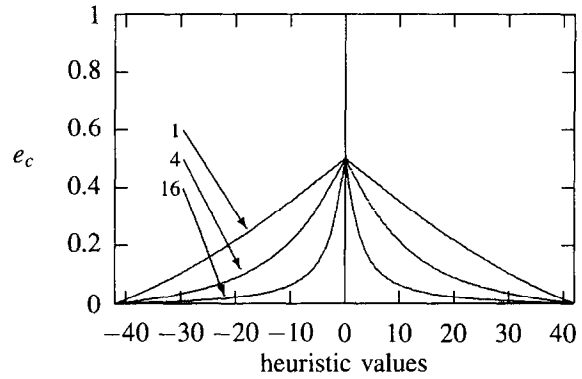


Fig. 1. Shape of arctan-based error functions $e_c$ for $c = 1, 4, 16$, and with $h_{\max} = 42$ for a heuristic evaluator of the material balance in chess counting the pieces' material values only.

**Definition 4.** The error probability $e(n)$ of the static value $h = f(n)$ of node $n$ assigned by the evaluation function $f$ is defined as

$$e(n) = P(u(f(n)) \neq TV(n)).$$

The probability of such an error is, however, not the same for all heuristic values. Experience with chess and checkers programs, for instance, has shown that very "strong" ("weak") positions with extremely high (low) heuristic values seldom have the actual status *LOSS* (*WIN*). The smaller the absolute value, the larger is the probability of incorrect evaluation. This clearly indicates that the error distribution is non-uniform in practice. We model this by the following conditions:

$$e(n_1) < e(n_2) \quad \text{if } h_1 = f(n_1) > h_2 = f(n_2) > 0,$$
$$e(n_1) < e(n_2) \quad \text{if } h_1 = f(n_1) < h_2 = f(n_2) < 0.$$

These conditions represent monotony requirements for error distributions according to our approach. For defining concrete error distributions that satisfy them, we make use of the "probability to win" distributions as defined by $w_c$ in Eq. (1). In fact, there is a strong relationship between the "probability to win" and the probability of error. Since such errors occur when a node with true value *WIN* (*LOSS*) is assigned a negative (positive) heuristic value, the probability of error $e_c(h)$ for negative heuristic values $h$ is defined by the "probability to win" $w_c(h)$. Analogously, the probability of error $e_c(h)$ for positive heuristic values $h$ is defined by the "probability to lose" $1 - w_c(h)$.

$$e_c(h) = \begin{cases} w_c(h), & h < 0, \\ 1 - w_c(h), & h > 0. \end{cases} \tag{2}$$

Since $w_c(-h) = 1 - w_c(h)$, $e_c(h)$ is symmetric. Fig. 1 shows the shape of three error functions $e_c$ parameterized by different values of $c$. The probability of error increases monotonically, reaches its maximum at $h = 0$, and decreases monotonically for positive

values. A high value of $c$ allows the specification of a "steep" error function (see Fig. 1). In such cases, even heuristic values of, e.g., $-10$ or $+10$ (out of the range $-42$ to $+42$) have a quite small probability of error. In fact, computer chess practice has shown that positions with such a material advantage of, e.g., two rooks (or similarly, a queen and a pawn) are very seldom lost.

Pearl's analysis [29] of his pathological model uses the assumption of independent and identically distributed (i.i.d.) true and heuristic values. Therefore, any specific distribution could be assumed for the heuristic values without changing the behavior of this model (as long as the identical distribution is used for each value independently). Consequently, the use of the described error function $e_c$ in this context does *not* change the pathological behavior of the model by Pearl. So, another assumption is yet needed to achieve the desired properties of our model.

### 3.2. *Dependency of heuristic values*

There is some agreement that the independence assumptions are the most critical ones in the pathological models. However, most of the previous work studied dependencies of the *true* values, and for these it is quite difficult to establish the correspondence to practice. But *how* do *heuristic* values depend on each other? In earlier work [15], we used *quiescence* to model such a dependency for *two* distinct heuristic values. In the context of *many* heuristic values, the assumption of independence of each other is even more unrealistic: it would imply that every value is possible at every node. However, in practice the changes in heuristic values are not completely random between successive positions. In particular, the values cannot change in one single step from one end of the scale to the other.

In fact, Fuller et al. [6] proposed an *incremental* model that was used by Newborn [28] and Knuth and Moore [16] for investigations about the pruning efficiency of the alpha-beta algorithm. Fuller et al. and Newborn randomly assign a value from the set $\{1, 2, \ldots, N\}$ to the branches and define the score (heuristic value) of a terminal node as the sum of the values of the branches on the path to this terminal node.

We adapt this dependency model slightly to better reflect the situation in practice. In the original model, values at different depths of the tree have a different meaning, since the values can only increase with increasing depth. We prefer to keep a constant limit point (i.e., 0) between "weak" and "strong" positions. Therefore, one should rather assign values from the set $\{-a, \ldots, +a\}$ to the branches ($a$ can be regarded as an augmentative value). Our experiments with this approach gave essentially the same results as those reported below, but we tried to obtain an even more realistic model. In chess programs, for instance, the static evaluation typically changes only in one direction according to which player is moving since, e.g., the capture of a piece changes the evaluation in favor of this player. Consequently, the static evaluation cannot decrease with *MAX* on move and increase with *MIN* on move.[5] Non-capture moves may also

[5] In *zugzwang* situations, the search may well discover that the backed-up value decreases with *MAX* on move because of being forced to move, but this does not contradict our approach to model the incremental dependency of heuristic values as assigned by the static evaluation function.

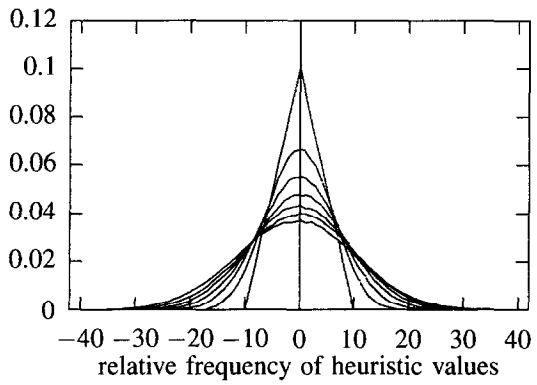


Fig. 2. Distribution of heuristic values at depths $d = 2, 4, \ldots, 14$. ($b = 2$, $h \in \{-42, \ldots, +42\}$, $a = 9$.)

leave the value unchanged. If 0 is the minimum change and $a$ denotes the maximum change in the evaluation between position $n$ and its child nodes $n_i$, this constraint can be formally written as

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

Now let us investigate how this dependency through incremental changes of values effects the distribution of the heuristic values at the search frontier. The incremental changes of the values are randomly assigned from the allowed range. Using this constraint the distribution of the heuristic values in a game tree cannot be uniform. Since the minimax value $MM_f^d(n)$ is computed using the heuristic values at the search frontier of depth $d$, the distribution of the heuristic values at this search frontier has an impact on the minimax value. Let us assume that the minimax value of a position with a heuristic value is computed that does not clearly indicate which side is in favor. (We denote such positions as *interesting* below.) Starting, e.g., with 0 as the heuristic value for the root node, most of the heuristic values found in the searched game tree will vary little from the root's value. A heuristic value of +42 can only occur after at least 10 plies (using $a = 9$).[6] Assuming, for instance, 0 as the heuristic value of the root node, a branching degree $b$ of 2, and possible heuristic values $\{-42, \ldots, +42\}$, Fig. 2 shows the distributions of heuristic values obtained in simulations at the search frontier for $d = 2, 4, \ldots, 14$. The "steepest" distribution is shown for $d = 2$, the "shallowest" for $d = 14$. On the whole, shallower distributions occur with increasing $d$. Since the heuristic value of the root node is 0, searches of depth $d = 2$ can only find heuristic values between $-9$ and $+9$ at the frontier. At depth $d = 4$, the range of the heuristic values is already $-18$ to $+18$, etc. Therefore, the distribution of the heuristic values at the search frontier has to become shallower with increasing search depth $d$.

[6] In chess, the value of a queen is usually considered to be 9. The case of check-mate is ignored.

Chess and checkers programs, for instance, utilize such a dependency of heuristic values for saving time by incremental computation of such a heuristic function. A small chess example below illustrates this. Since this kind of dependency occurs in such programs, also its effect on the distribution of the heuristic values at the search frontier occurs, which we observed ourselves in traces of the search trees generated by our own computer chess program.

Note, that this is the first use of such a dependency assumption that relates the *heuristic* values for the investigation of minimax pathology/benefits issues. Although the N-game model by Nau [27] was inspired by the same idea, it relates the *true* values during the construction of a specific game board. In our model, the true values are also dependent since they are related to the heuristic values by the error probabilities and since the heuristic values depend on each other. However, this dependency of the true values is only *indirect*, and in our model the errors are *not* uniformly distributed. We prefer to model the dependency of the heuristic values as described, since it is quite obvious in practice.

## 4. Experiment design

In order to examine the behavior of our model, we performed experiments in the form of simulation studies similar to those described in [15]. Before we discuss the results, let us describe the experiment design.

In essence, synthetic game trees satisfying the model assumptions were generated first. Then minimax searches of different depths were performed in the generated game trees. These searches computed minimax values $MM_f^d(n)$ for all the possible depths $d$ in such a tree.[7]

The goal of performing these searches was to gather data for the static and the dynamic evaluation errors as well as errors of move decisions. The emphasis of these simulation studies was on how the various errors change with increasing search depth.

We restricted the searches to *interesting* situations at the root. Intuitively, we define a situation as interesting that is represented by a node with an inconclusive evaluation. The precise meaning of *interesting* depends on the error probability $e$ as well as on the range of heuristic values and is given below for the respective experiments.

The reason for restricting the searches is that deeper searches are really useful when the root and the nodes encountered during shallow searches have inconclusive evaluations with a high error probability. This can be observed in practice as well as explained according to our model. In case of an $e_c$-type error distribution (illustrated, e.g., in Fig. 1), searches starting from a value in the vicinity of 0 encounter heuristic values with a relatively high probability of error at shallow search depths. The actual status of these nodes could very likely be *WIN* as well as *LOSS*. In order to clarify the

[7] 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; see for instance [16] for a description of the alpha-beta algorithm. This pruning complicates the reproducibility of the trees, so that we stored them in memory. A more recent approach to random tree generation during the search is described in [19, Section 4].

Table 1
Overview of our proposed assumptions and the corresponding models

| Assumptions used | | Dependency of heuristic values | |
|---|---|---|---|
| | | yes | no |
| Non-uniformity of error distribution | yes | our complete model | Pearl's basic model |
| | no | "uniform" model | Pearl's basic model |

situation, deeper minimax searches can be used. In contrast, if a minimax search starts, for instance, at a node with a heuristic value of −42 (at the very left of the scale), then it is unlikely that the minimax value is going to be positive. Since such a negative heuristic value also has a low probability of error (according to the non-uniform distribution of errors), this node can reasonably be considered to be *LOSS* independent of the search depth $d$. In this case deeper minimax searches would just confirm the result of a shallow search.

Primarily, we were interested in the results of those experiments with trees according to our complete model that includes both of our additional assumptions. Still, there was the important question whether both of these assumptions are necessary to achieve the results that compare favorably with the observations from practice. In order to answer this question, it may seem that experiments with either and both assumptions removed would have to be made. However, not all of these cases had to be dealt with in the experiments, since some of them are covered by already existing theoretical results. Table 1 provides an overview of our proposed assumptions and the corresponding models. Both cases with no dependency of heuristic values are covered by the basic (pathological) model of Pearl [29] due to its i.i.d. assumption as discussed above. So, we designed the experiments for our complete model, and for a "uniform" model with only the assumption of non-uniformity of error distribution removed. In the "uniform" model, all the heuristic values have the same error probability.

### *4.1. Error definitions*

In order to examine the behavior of the model, we define how the behavior can be measured in terms of errors. The principal question is whether the probability of (static) error $e$ made by the evaluation function (as already defined above) is increased or decreased by backing up the heuristic estimates through several levels of the tree via minimaxing. We characterize these *dynamic* errors made by the minimax value of a certain search depth in estimating the true value as follows.[8]

**Definition 5.** The error probability $e^d(n)$ of the minimax value $MM_f^d(n)$ of node $n$ from a search to depth $d$ using the evaluation function $f$ is defined as

$$e^d(n) = P(u(MM_f^d(n)) \neq TV(n)).$$

[8] This definition is a generalized version of Definition 4.2 in [33].

In our model a closer look at the probability of *static* error is also of interest, since it changes with the search depth although the same evaluation function $f$ is used at all levels of the tree, and although its general error is independent of the depth. While $e$ of Definition 4 characterizes this *general* error made by the evaluation function, we are also interested in the specific error of evaluating those nodes statically whose values are actually backed-up through minimaxing.

**Definition 6.** The error probability $e_s^d$ of the static value $h = f(n_d)$ (assigned by the evaluation function $f$) of node $n_d$ at the search frontier with depth $d$ from node $n$ is defined as

$$e_s^d(n) = P(u(f(n_d)) \neq TV(n_d) \mid n_d \text{ is a successor of depth } d \text{ of node } n).$$

### 4.2. Tree generation

Generally, every stochastic event in the tree is simulated by a call to a *pseudo-random number generator*, parameterized independently of the relative frequencies achieved earlier in the tree generation process.[9]

The game trees of depth $d_g$ are generated top-down beginning at the root node by a recursive procedure. Heuristic and true values are assigned to the child nodes $n_i$ of a node $n$ such that *first* Eq. (3) constraining the heuristic values and *then* the game-theoretic relationship constraining the true values are established.[10] The heuristic values of the child nodes $n_i$ are randomly chosen according to Eq. (3). Once given the heuristic values, the true values can be assigned to the nodes $n_i$ with the probability of error $e_c$ (or uniform probability of error) under the constraint of the game-theoretic relationship.

## 5. Results

As indicated above, we gathered data about the static and dynamic evaluation errors, as well as the quality of move decisions. The emphasis was to see if and how they change with increasing search depth. In addition, we studied the influence of the number of distinct values—the granularity—on the results of minimaxing.

### 5.1. Evaluation errors

Fig. 3 shows the probability of *static* evaluation error $e_s^d$ for different search depths $d$ (actually we only show the data for each move cycle, i.e., two plies) in our com-

[9] The ones used are **nrand48** and **erand48** available under UNIX. UNIX is a registered trademark of AT&T.

[10] Due to the shape of the function $e_c$ (see Eq. (2)), 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*.

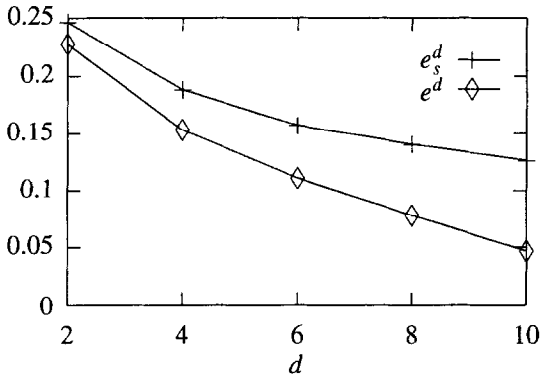


Fig. 3. Errors in our complete model. ($d_g = 15$, $b = 2$, $h \in \{-42, \ldots, -1, +1, \ldots, +42\}$, $a = 9$, $e \approx 0.15$.)

plete model. The data were gathered during searches in *interesting* situations. To define interesting situations for these experiments with $h_{\max} = 42$, we used for the values of root nodes the sets $\{-1, +1\}$ and a general static evaluation error $e \approx 0.15$, alternatively $\{-5, \ldots, +5\}$ and $e \approx 0.1$, as well as $\{-9, \ldots, +9\}$ and $e \approx 0.1$. For Figs. 3 and 4, the first of these definitions was used, but we found analogous behavior for the other ones. On the whole, the value of $e_s^d$ decreases with increasing $d$—the quality of static evaluation improves with increasing search depth—although the same evaluation function is used at all levels, and although its general error is independent of the depth. We call this behavior *improved evaluation quality*.

This improvement can be explained in the following way. Since searches start at interesting positions, large heuristic values can only be found at searches with a large $d$. Therefore, the relative frequency of large heuristic values increases with the search depth $d$ (see also Fig. 2). Using $e_c$ according to our complete model, these heuristic values $h$ have smaller error probabilities. Therefore, the probability of static evaluation error $e_s^d$ has to decrease with increasing $d$—deeper searches more often find larger heuristic values with smaller error probabilities.

In contrast, assuming uniform distribution of the static error as in the "uniform" model according to Table 1, $e_s^d$ has to remain constant. Deeper minimax searches cannot find heuristic values with a lower error probability because all of them have the *same* error probability.

Figs. 3 and 4 show the impact of the different static error distributions ($e_c$ or uniform, respectively) on the *dynamic* error $e^d$. In the case of our complete model that uses $e_c$, Fig. 3 shows a sharp absolute and relative reduction of $e^d$ in the beginning, which dampens with increasing $d$. The behavior of $e^d$ is clearly not pathological but strongly beneficial. This kind of behavior can be observed in practice as discussed below in regard to move decisions. Fig. 4 shows the behavior for $e^d$ in the case of the "uniform" model that uses a *uniform* static error distribution—starting at $d = 8$, $e^d$ even shows a slight tendency to increase (i.e., pathological behavior). The *absolute* values of Fig. 3 and Fig. 4 are not comparable. In *interesting* situations, searches with an

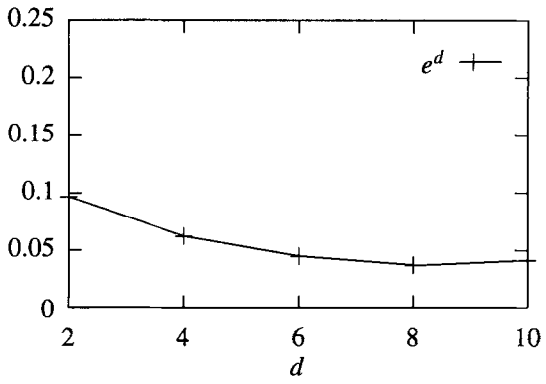


Fig. 4. Errors in the "uniform" model. ($d_g = 15$, $b = 2$, $h \in \{-42, \ldots, -1, +1, \ldots, +42\}$, $a = 9$, $e \approx 0.15$.)

$e_c$-type error distribution (summarized, e.g., in Fig. 3) encounter heuristic values with a relatively high probability of error at shallow search depths, whereas for a uniform error distribution even heuristic values near 0 have a *low* probability of error. Therefore, $e^d$ of Fig. 3 has to be larger than $e^d$ of Fig. 4 for shallow search depths. In general, the improvement of the dynamic and static evaluation quality in our complete multivalued model with both assumptions—non-uniformity of error distribution and dependency of heuristic values—is much more pronounced than in a model which disregards the assumption of non-uniformity of error distribution in favor of using a *uniform* error distribution.

According to the assumptions in the model of Pearl [29] the quality of the evaluation function has to increase per move cycle by more than 50 percent just to combat pathology. In our model, the static evaluation quality $e_s^d$ of Fig. 3 increases between 10.4 percent (comparing $d = 8$ to $d = 10$) and 23.5 percent (comparing $d = 2$ to $d = 4$) per move cycle and we achieve a significant improvement of the dynamic evaluation quality $e^d$ with increasing search depth $d$. Due to the incremental dependency of the heuristic values in our model, the assumptions of Pearl's analysis are not valid here. Therefore, this behavior does not contradict Pearl's results.

Experiments with different values of $h_{\max}$ and $a$ showed the same qualitative results. Therefore, our results are not specific to chess but are generalizable to situations in kalah and checkers by using appropriate values for $h_{\max}$, $a$ and $c$.

### 5.2. *Move decisions*

Now let us see how the errors characterized by $e^d$ influence the move decisions. A "player" based on minimaxing chooses one of possibly several moves leading to the maximum of those values that are backed-up from all the subtrees rooted in the successor positions. So, the errors of dynamically evaluating positions using minimaxing induce errors of move decisions.

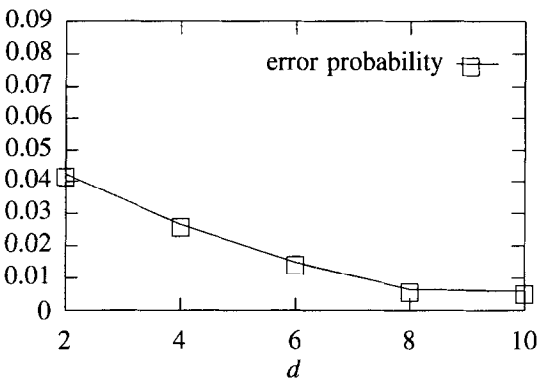


Fig. 5. Quality of move decisions in our complete model. ($d_g = 15$, $b = 2$, $h \in \{-42, \ldots, -1, +1, \ldots, +42\}$, $a = 9$, $e \approx 0.15$.)

Analogously to Pearl [29, p.431], we are interested in the probability of error of such move decisions. Assuming a game position with two successors, a *WIN* and a *LOSS*, an error is induced when the *WIN* position obtains a lower (dynamic) evaluation than its *LOSS* sibling. In case of equal evaluations of *WIN* and *LOSS* and arbitrary tie-breaking (as usual), there is a 50 percent chance of inducing a decision error.

As would be expected from the results presented above, in our model the quality of move decisions made by using minimaxing improves with increasing search depth. Fig. 5 illustrates this for the same experiments as shown above in Fig. 3. Much as the error probability $e^d$ of dynamic evaluation is reduced with increasing search depth, the probability of wrong move decisions is reduced. Similarly to the improvement of $e^d$ with increasing search depth, the improvements in minimaxing's move decisions also dampen. While this phenomenon is qualitatively consistent with computer chess and checkers practice, our data cannot be compared quantitatively with that in a specific game, since for chess and checkers the true values are mostly unknown. Still, we can compare our results with those from experiments on the relative playing strength. For instance, Condon and Thompson [5] performed an experiment on the influence of search depth on the playing strength of full-width searching chess programs. The results gained indicate that increasing the search depth by an additional ply of search (with an identical static evaluation function) leads to a significant increase in the score achieved against the shallower searching program. This improvement also dampened for larger search depths. More precisely, the performance rating improved by 335 points through increasing the full-width search depth from 4 to 5, and this improvement dampened strictly monotonically to 120 from depth 8 to 9 [5, Fig. 9.5]. Junghanns and Schaeffer [9] report on recent investigations that show this phenomenon in a new way for real programs in games like chess, checkers and Othello.

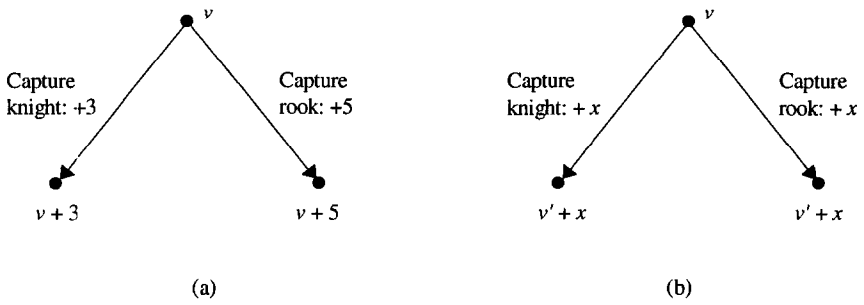


Fig. 6. A chess example illustrating a problem related to granularity.

### 5.3. Granularity

Since the main difference of our model proposed in this paper to other models is the particular use of multivalued evaluation functions, the question may arise, how many distinct values should be used. In other words, the relation of granularity to performance is of interest. Intuitively, it may seem the finer-grained the evaluations the better. In the following, we investigate this issue based on our experiments with our model and its relation to real programs for games like chess.

We performed experiments with different numbers of distinct values. In order not to change anything else, we kept the same proportions between $h_{\max}$, $a$ and the parameter for "interestingness", and used proportionally adapted error distributions. For instance, in chess a rook is usually evaluated as 5 pawn units. So, assuming the value of a pawn is 1 point, the value of a rook is 5 points. Assuming, however, that the value of a pawn is 100 points, the value of a rook must be 500 points, so that no change about this "knowledge" is introduced. The value of 500 points must also have the same error probability in these experiments as the value of 5 points in the other.

The results of these experiments with different numbers of distinct values were the *same* as those illustrated and explained above. At first, these results may be quite surprising, since they may seem to contradict the intuition stated above. So, let us have a closer look at the reasons for these experimental results. As explained above, the important effect according to our model is *improved evaluation quality*. It arises because the relative frequency of (relatively) large heuristic values increases with the search depth $d$ (see again Fig. 2), and because these values have smaller error probabilities assigned than smaller heuristic values (see again Fig. 1). Since in these experiments the relevant proportions were constant, the same effect arises for different granularities, both qualitatively and quantitatively.

Still, the question may arise what happens when there are too few distinct values, so that the proportions cannot be constant. For the illustration of this intricate case we prefer to use a simple chess example. For its understanding, no special knowledge about chess is required beside the relative values of knight and rook, which are usually 3 and 5 pawn units, respectively. Fig. 6(a) shows two capture moves from a position with

some static value $v$. Let us assume that only the material values are considered here, so that the move to the left leads to a position with a static value $v + 3$, since it captures a knight.[11] Analogously, the move to the right leads to a position with a static value $v + 5$, since it captures a rook. Obviously, the values of the resulting positions can be distinguished, so that apart from the given error probabilities the "right" move will be chosen (either for minimax back-up or for being played). In contrast, Fig. 6(b) shows the situation when the values of both a knight and a rook are the same, e.g., $x = 2$. In this case, the capturing moves have the same incremental value. Consequently, also the values of the resulting positions are the same and therefore indistinguishable. When such a value is backed up, it does not reflect the difference between these moves and their resulting positions. A move decision in such cases is usually made arbitrarily (see also above), i.e., an additional error is introduced into such a move decision.

This example shows what can happen when too few values are used compared to the *knowledge* available—in this case knowledge about relative values of chess pieces. Since the values determine a partial order of positions, using too few values reduces the ability to discriminate between positions and leads to a higher rate of random move decisions. The lower the granularity, the more likely such cases (as in the example) occur. For games with two true values, the extreme is to use just two heuristic values. Compare this also with the results of Adapted-Merlin given in the Appendix. Because of having a third true value—*DRAW*—in chess, there are also three heuristic values. In our model, we achieve such cases by mapping to fewer values, which leads to identical values of a node's children (much as in the chess example above) and therefore to a higher rate of random move decisions. Once the model reduces to the case of only two heuristic values, the effect of improved evaluation quality cannot be achieved any more.

In theory, the use of infinitely many distinct values or even real numbers could resolve the issue of random move decisions. In practice, however, there seems to be a certain limit for the usefulness of a larger numbers of values. In computer chess, for instance, typically a grain size around $1/100$ (or for technical reasons $1/128$) pawn unit appears to be used, and the checkers program Chinook uses a grain size of $1/100$th of a pawn.[12] This grain size is related to the available *knowledge*. While the best chess and checkers programs have knowledge about certain positional aspects, their evaluations are dominated by the material balance. A larger number of distinct values than needed to express the available knowledge is of no use. Our experiments with different numbers of distinct values confirm this.

When more distinct values arise during search than useful relative to the given knowledge, this is even (slightly) harmful. More distinct values also mean less alpha-beta pruning, since a cutoff is normally achieved also for equal values: if more often different values arise, fewer cutoffs occur and therefore more nodes have to be searched. Although this is only a second-order effect in our context, it suggests not to use more distinct values than useful to express the knowledge available.

[11] Compare this incremental computation with the approach chosen for modeling the dependency of heuristic values.

[12] According to private communication with Monty Newborn and Jonathan Schaeffer.

## 6. Related work

A fair bit of attention has been focused on the theoretical problem of the effects of minimaxing under various conditions. Since critical overviews can be found in [13,15], our summary can be rather cursory. Nau [22,23,25] showed 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. He called such behavior *pathological*. Schrüfer [34, Section 1] modified Nau's model, distinguishing two different error parameters for "overestimating" and "underestimating", respectively.

In later work Nau [24] investigated a class of "real" games, called Pearl's games or P-games, in which pathology actually occurs. Similar classes of games, called N-games and G-games, were shown not to be pathological [26,27]. Essentially the same findings were reported independently by Beal [2,3] and Bratko and Gams [4]. A model of Schrüfer [34, Section 2] also relates the effects of deeper minimaxing more generally to the distribution of the true values. Michon [21] found non-pathological behavior for games with "inert" structure, despite the assumption of independent terminal values.

While all this work provided some insight into minimax pathology, there was no success in explaining the strong improvements with increasing search depth observed in practice. A common argument investigated by Pearl [29] is "the avoidance of traps". The model is oversimplified, considering only actual terminal nodes (i.e., check-mates and stalemates) at all levels of the game tree as "traps". Instead of such traps—but with the same consequences—Abramson [1] postulates that the evaluation function must be able to recognize more and more forced wins (evaluated *without* error) as search deepens, to avoid pathology.

In [13,15,33] we described and studied a rather complicated class of models based on the concept of *quiescence* (i.e., evaluations of stable positions are more reliable). Although we used only a binary evaluation function, results corresponding to practice could be obtained. However, no explanation for the benefits of using a multivalued evaluation function was given.

Another argument in attempting to explain the practical observations is that of "improved visibility", which was simply interpreted and modeled by Pearl [29] in the sense that the accuracy of the static evaluation function improves as the game proceeds. Mathematical analysis of this model shows that a very strong improvement (>50 percent per move cycle) is necessary to combat pathology, but this is based on the unrealistic assumption of independence of values. In the game of chess, no such improvement of accuracy can be seen when considering the static evaluation functions used in actual programs. On the contrary, there is consensus among computer chess experts that the endgame evaluators are in general worse than those for the midgame—but within the horizon of one search the quality of an evaluator remains nearly constant. Hence, at least in this domain (where minimaxing is very successful) improved visibility should be interpreted differently.

While this idea of improved visibility is the one closest to our notion of *improved evaluation quality*, these should be clearly distinguished. Pearl assumed that the accuracy of the static evaluation function itself improves very strongly as the game proceeds. In contrast, we assume an evaluation function with an error *independent* of the depth.

However, the evaluation quality *indirectly* improves during the search, based on the assumptions of non-uniformity of error distribution and dependency of heuristic values. Due to this dependency, it is not necessary in our model to have such strong improvements in the static evaluation quality.

As discussed above, the N-game model by Nau [27] was also inspired by the idea of incremental dependencies. However, in contrast to our approach it relates the *true* values during the construction of a specific game board. Our model focuses on dependencies between *heuristic* values, analogous to the models of Fuller et al. [6], Knuth and Moore [16] and Newborn [28] as used in a different context. (In fact, our model is the first use of this approach for investigating minimax pathology/benefits issues.) Of course, in our model a dependency of the true values is induced indirectly via the (non-uniformly distributed) error probabilities. We prefer to model the dependency of the heuristic values as described, since it is quite obvious in practice.

Moreover, many of the other theoretical analyses were based on models with only *two* heuristic values. Those with *multivalued* evaluation functions did *not* have any additional assumptions regarding the error distribution or dependency of the heuristic values. Studies by Bratko and Gams [4] and by Pearl [29] even compared multivalued to corresponding binary models and yielded the same results. In fact, the additional values did not carry any semantic content in these models. In contrast, our model reflects the very reason for using multiple values in practice: they serve as a means for representing more fine-grained differences among positions, where the higher-rated ones are typically better and have a higher probability to win associated with them. This is directly related to having a smaller error of incorrect evaluation.

The results shown by our new model correspond well to the observations in practice, and we are not aware of any unrealistic (or even pathological) effect. Therefore, we claim that our new model is better than previous ones at explaining real phenomena. Both its assumptions and its results are closer to minimaxing practice.

## 7. Conclusion

Until now, the dramatic benefits of using *minimaxing* for deeper and deeper searches in game-playing programs (as observed, e.g., in computer chess and checkers practice) have not yet been fully explained. Partly, the reason appears to be that binary evaluation functions were primarily used in the models. The role of multivalued functions was not fully investigated yet, i.e., no special meaning was assigned to the various values.

Our approach uses two fundamental assumptions regarding multivalued evaluation functions—non-uniformity of error distribution and dependency of heuristic values. We explained the observed correspondence of these assumptions to practice. Moreover, their basic characteristics already existed in the literature, but in different contexts and for different purposes.

Also in correspondence with practice, our multivalued model exhibits strong beneficial effects of minimaxing. This is primarily based on *improved evaluation quality* with increasing search depth although the same evaluation function is used at all levels of the tree, and although its general error is independent of the depth. Essentially, with

increasing search depth the evaluation function is more frequently used on positions that a multivalued function with the assumed properties can more reliably evaluate. This effect together with their ability to discriminate between positions leads to the benefits of using a *multivalued* evaluation function for minimaxing. Still, a larger number of distinct values than needed to express the available knowledge is of no use.

In summary, this is the first work showing the reason why the use of a multivalued evaluation function is important. Its assumptions as well as its results correspond very closely to practice. Since it shows why the results of deeper searches become much more reliable, this work also provides an explanation why searching deeper using minimaxing has been that successful in practice. While the starting point for our approach was the modeling of chess and checkers practice, our models simply exploit certain properties of the evaluation function. Therefore, we believe that the model is relevant for all games where the evaluation functions have these properties. Consequently, our approach may be the solution to the long-standing problem of explaining the strong benefits of deeper and deeper search using minimaxing.

## Acknowledgments

First, we thank Helmut Horacek and Jonathan Schaeffer for discussions on this topic. We appreciate the help by Tim Newman in improving the style and English of this paper, and we acknowledge the useful comments on earlier drafts by Wilhelm Barth, Helmut Horacek, Stefan Kramer, Jonathan Schaeffer and Weixiong Zhang. Finally, we would like to thank the anonymous reviewers for challenging us to study some important aspects in more depth.

## Appendix A. Merlin experiment

In the context of this research, we were interested to see how much a real tournament program degrades when it can only use the same number of heuristic values as true values in the game of question, i.e., three in the case of *WIN*, *LOSS*, and *DRAW*. Since we had the former chess tournament program Merlin [13] at our disposal, we decided to adapt it accordingly (more precisely, the 1989 World Championship version).

For backing-up according to the minimax rule, the heuristic values $h$ of Merlin's evaluation function were mapped as follows:

$$h < 0 \rightarrow -1, \text{ estimating } LOSS$$
$$h = 0 \rightarrow 0, \text{ estimating } DRAW$$
$$h > 0 \rightarrow +1, \text{ estimating } WIN$$

[13] Merlin is a collaborate effort of the second author together with Helmut Horacek and Marcus Wagner. It played some major computer tournaments; for instance, it tied for tenth out of 22 participants at the World Computer Chess Championship in 1983 and for sixth out of 24 participants at the same tournament in 1989.

It is important to note, however, that the original heuristic values were still used for guiding the search to variable depth [12], and in particular for Merlin's elaborate *quiescence search* [10,11]. That is, the available knowledge about fine-grained evaluations was still partly used, but not for minimaxing. In this paper, we refer to this version as *Adapted-Merlin*.

Essentially, there were two possibilities to measure the difference in the programs' strength: having Merlin play a match against Adapted-Merlin, or using a benchmark test. However, the cost of complete tournament games would have exhausted our resources (CPU time on a fast mainframe computer). Moreover, the data on a widely used set of positions (the Bratko–Kopec test[14] [17]) are available for all the participants of the World Computer Chess Championship in 1989 (including Merlin) [20]. Therefore, we decided to run Adapted-Merlin on this set, providing the same amount of resources as used by Merlin in producing the result reported in the literature [20].

The summarized data of this run are as follows: Adapted-Merlin scores 5 points out of 12 on the $T$ set and 1 point out of 12 on the $L$ set, resulting in an overall score of 6 out of 24 possible points. In comparison, the worst result of the Championship participants in 1989 was 10. However, Merlin achieved a score of 16 $(10 + 6)$. In fact, Adapted-Merlin performed clearly worse, which is highly statistically significant according to the sign test.

In order to derive more insights from this experiment, let us sketch some observations:

- Even in the positions where Merlin did not find the correct move, it selected a reasonable move. In contrast, Adapted-Merlin has a tendency to blunder. This can be explained as follows: if this program "thinks" it loses (i.e., it can only achieve a score of $-1$), every move is acceptable to the program, and the first one is selected.
- There were many failures of Adapted-Merlin in $T$ positions, although the search depth was clearly sufficient for having all the critical variations included in the tree searched. The reason is symmetric to the one above: if the program "thinks" it wins (i.e., it achieves a score of $+1$), every move leading to this value is considered to be as good as the others, and therefore the first one found is selected.
- The main variation / principal continuation (i.e., the sequence of moves along which the minimax value is backed up) contains many blunders in Adapted-Merlin. This means that the reasons for bad move selections given above apply all over the search tree.
- Adapted-Merlin has of course *no* special problem in positions where check-mate can be found (the first position of the $T$ set is of this type).
- We could not find any indication of *pathological* behavior even in the adapted version. That is, at least in this test set there was no single case where the deeper search iterations[15] selected a worse move than the shallower ones.

[14] There are 24 positions: 12 tactical, denoted by $T$, and 12 "levers" denoted by $L$. The latter contain positional (or "strategic") themes related to critical pawn moves.

[15] Like most tournament programs, Merlin uses *iterative deepening*. The usefulness of this paradigm in minimax search is reviewed in [14], and a well-known heuristic search procedure based on iterative deepening can be found in [18].

- Adapted-Merlin searched fewer nodes in each iteration than Merlin. Of course, cutoffs can be achieved more often with this reduced set of heuristic values.
- While the result on the $L$ positions was particularly bad, good positional moves can also be found by the adapted version under certain circumstances: apart from "lucky" move ordering, such a move should have a different heuristic value than the remaining moves. This occurred in the one $L$ position that Adapted-Merlin solved to our surprise.

In summary, even if sufficient knowledge is available in a program for selecting the right move, the adaptation to use just three distinct heuristic values can lead to the loss of this knowledge in the course of back-up. Due to the significant result on the benchmark and the described observations, it is obvious that Adapted-Merlin would be decisively defeated in a match by Merlin. The tendency to blunder in the way observed has very bad consequences in playing games like chess. Therefore, we conclude that the benefits of searches using minimaxing are much more pronounced with multiple heuristic values.

## References

[1] B. Abramson, An explanation of and cure for minimax pathology, in: L.N. Kanal, J.F. Lemmer (Eds.), Uncertainty in Artificial Intelligence, North-Holland, Amsterdam, 1986, pp. 495–504.
[2] D.F. Beal, An analysis of minimax, in: M.R.B. Clarke (Ed.), Advances in Computer Chess, Vol. 2, Edinburgh University Press, 1980, pp. 103–109.
[3] D.F. Beal, Benefits of minimax search, in: M.R.B. Clarke (Ed.), Advances in Computer Chess, Vol. 3, Pergamon Press, Oxford, 1982, pp. 17–24.
[4] I. Bratko, M. Gams, Error analysis of the minimax principle, in: M.R.B. Clarke (Ed.), Advances in Computer Chess, Vol. 3, Pergamon Press, Oxford, 1982, pp. 1–15.
[5] J.H. Condon, K. Thompson, Belle, in: P.W. Frey (Ed.), Chess Skill in Man and Machine, 2nd ed., Springer, Berlin, 1983, pp. 201–210.
[6] S.H. Fuller, J.G. Gaschnig, J.J. Gillogly, An analysis of the alpha-beta pruning algorithm, Technical Report, Carnegie Mellon University, Pittsburgh, PA, 1973.
[7] H. Horacek, Reasoning with uncertainty in computer chess, Artificial Intelligence 43 (1990) 37–56.
[8] H. Horacek, H. Kaindl, M. Wagner, Probabilities in game-playing: possible meanings and applications, in: Proceedings 3rd Austrian Meeting on Artificial Intelligence, Vienna, Springer, Berlin, 1987, pp. 12–23.
[9] A. Junghanns, J. Schaeffer, Search versus knowledge in game-playing programs revisited, in: Proceedings IJCAI-97, Nagoya, Japan, 1997.
[10] H. Kaindl, Dynamic control of the quiescence search in computer chess, in: Proceedings EMCSR-82, Vienna, North-Holland, Amsterdam, 1982, pp. 973–978,
[11] H. Kaindl, Quiescence search in computer chess, in: M.A. Bramer (Ed.), Game-Playing (special issue), SIGART Newsletter 80 (1982) 124–131; Reprinted in: Computer Game-Playing: Theory and Practice, Ellis Horwood, Chichester, UK, 1983, pp. 39–52.
[12] H. Kaindl, Searching to variable depth in computer chess, in: Proceedings IJCAI-83, Karlsruhe, Germany, 1983, pp. 760–762.
[13] H. Kaindl, Minimaxing: theory and practice, AI Magazine 9 (3) (1988) 69–76.
[14] H. Kaindl, Tree searching algorithms, in: T.A. Marsland, J. Schaeffer (Eds.), Computers, Chess, and Cognition, Springer, New York, 1990, pp. 133–158.
[15] H. Kaindl, A. Scheucher, Reasons for the effects of bounded look-ahead search, IEEE Trans. Systems Man Cybernet. 22 (5) (1992) 992–1007.
[16] D.E. Knuth, R.W. Moore, An analysis of alpha-beta pruning, Artificial Intelligence 6 (4) (1975) 293–326.

[17] D. Kopec, I. Bratko, The Bratko–Kopec experiment: a comparison of human and computer performance in chess, in: M.R.B. Clarke (Ed.), Advances in Computer Chess, Vol. 3, Pergamon Press, Oxford, 1982, pp. 57–72.
[18] R.E. Korf, Depth-first iterative-deepening: an optimal admissible tree search, Artificial Intelligence 27 (1) (1985) 97–109.
[19] R.E. Korf, D.M. Chickering, Best-first minimax search, Artificial Intelligence 84 (1996) 299–337.
[20] T.A. Marsland, The Bratko–Kopec test revisited, in: T.A. Marsland, J. Schaeffer (Eds.), Computers, Chess, and Cognition, Springer, New York, 1990, pp. 217–223.
[21] G. Michon, Recursive random games: a probabilistic model for perfect information games, Ph.D. Thesis, University of California, Los Angeles, CA, 1983.
[22] D.S. Nau, Quality of decision versus depth of search on game trees, Ph.D. Thesis, Duke University, Durham, NC, 1979.
[23] D.S. Nau, Pathology on game trees: a summary of results, in: Proceedings AAAI-80, Stanford, CA, 1980, pp. 102–104.
[24] D.S. Nau, An investigation of the causes of pathology in games, Artificial Intelligence 19 (3) (1982) 257–278.
[25] D.S. Nau, Decision quality as a function of search depth on game trees, J. ACM 30 (4) (1983) 687–708.
[26] D.S. Nau, On game graph structure and its influence on pathology, Internat. J. Comput. Inform. Sci. 12 (6) (1983) 367–383.
[27] D.S. Nau, Pathology on game trees revisited, and an alternative to minimaxing, Artificial Intelligence 21 (1–2) (1983) 221–244.
[28] M.M. Newborn, The efficiency of the alpha-beta search on trees with branch-dependent terminal node scores, Artificial Intelligence 8 (1977) 137–153.
[29] J. Pearl, On the nature of pathology in game searching, Artificial Intelligence 20 (4) (1983) 427–453.
[30] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Reading, MA, 1984.
[31] J. Schaeffer, R. Lake, P. Lu, M. Bryant, CHINOOK: the world man–machine checkers champion, AI Magazine 17 (1) (1996) 21–29.
[32] A. Scheucher, Untersuchung von Minimaxing in einem mehrwertigen Modell, Diplomarbeit, Wirtschaftsuniversität Wien, 1992.
[33] A. Scheucher, H. Kaindl, The reason for the benefits of minimax search, in: Proceedings IJCAI-89, Detroit, MI, 1989, pp. 322–327.
[34] G. Schrüfer, Presence and absence of pathology on game trees, in: D.F. Beal (Ed.), Advances in Computer Chess, Vol. 4, Pergamon Press, Oxford, 1986, pp. 101–112.