Bias and pathology in minimax search

A. Sadikov[1], I. Bratko, I. Kononenko

University of Ljubljana, Ljubljana, Slovenia

Abstract

This article presents the results of experiments designed to gain insight into the effect of the minimax algorithm on the error of a heuristic evaluation function. Two types of effect of minimax are considered: (a) *evaluation* accuracy (are the minimax backed-up values more accurate than the heuristic values themselves?), and (b) *decision* accuracy (are moves played by deeper minimax search better than those by shallower search?). The experiments were performed in the KRK chess endgame and in randomly generated game trees. The results show that, counter-intuitively, evaluation accuracy may decline with search depth, whereas at the same time decision accuracy improves with depth. In the article, this is explained by the fact that minimax in combination with a noisy evaluation function introduces a bias into the backed-up evaluations, which masks the evaluation effectiveness of minimax, but this bias still permits decision accuracy to improve with depth. This observed behaviour of minimax in the KRK endgame is discussed in the light of previous studies of pathology in minimax. It is shown that explaining the behaviour of minimax in an actual chess endgame in terms of previously known results requires special care.

Keywords: minimax principle, evaluation-function quality, bias, minimax pathology, KRK chess endgame

1 Introduction

Over twenty years ago Beal [4][2] and Nau [11,12] independently of one another set out to analyze whether and why values backed up from minimax search are more trustworthy than the heuristic values themselves. Beal constructed a simple mathematical model to analyze the minimax algorithm. To his surprise, the analysis of the model showed that the backed-up values were actually somewhat less trustworthy than the heuristic values themselves. He then wrote: "This result is disappointing. It was hoped that the analysis would show that the probability of error reduced with backing-up." A couple of years later two articles [5,8] simultaneously conducted further analysis into why minimax does yield good results in practical game-playing while apparently backed-up values seem less reliable; both articles independently reached the same conclusion. They argued that the true values of sibling nodes in a game tree are not independent of one another. This clustering of similar values is a major feature in practical games and it was this phenomenon that Beal's original mathematical model (1980) did not account for. The problem with the minimax paradigm under the assumption of independence of sibling values was also found by Nau [11,12,13,14], who called this a *search-depth pathology* in game trees.

[1] University of Ljubljana, Faculty of Computer and Information Science, Tržaška 25, 1000 Ljubljana, Slovenia. Email: aleksander.sadikov@fri.uni-lj.si.

[2] Beal [4] is the book version of a paper presented at the Advances in Computer Chess 2 conference which was held at Edinburgh University in April 1978.

In a simulation, Nau [13] introduced strong dependencies between sibling nodes and discovered that this can cause search-depth pathology to disappear.

It should be noted that the effectiveness of minimax search could be studied in two senses: (a) Do minimax backed-up position evaluations become more accurate with deeper search? (b) Are decisions, that is moves played, better when they are based on deeper search? We will refer to these two aspects as *evaluation* accuracy and *decision* accuracy of minimax, respectively. Nau [11] was primarily concerned with decision accuracy, while Beal [5], and Bratko and Gams [8] were concerned with evaluation accuracy. Accordingly, we will distinguish between two types of pathology: evaluation pathology and decision pathology. Although the two types of pathology are obviously related, they may not completely coincide.

Pearl [15] and later Abramson [1] partly disagreed with the conclusion reached by Beal, Bratko and Gams, and Nau, and claimed that while strong dependencies between sibling nodes in a game tree can eliminate the pathology, practical games such as chess do not possess dependencies of sufficient strength. Pearl pointed out that few chess positions are so strong that they cannot be spoiled abruptly if one really tries hard to do so. He concluded that the success of minimax in game-playing programs is "based on the fact that common games do not possess a uniform structure, but are riddled with early terminal positions, colloquially named blunders, pitfalls or traps. Close ancestors of such traps carry more reliable evaluations than the rest of the nodes, and when more of these ancestors are exposed by the search, the decisions become more valid." Moreover, Schrüfer [16] and Althöfer [2] did some further analysis of pathology in game trees. Especially interesting is Schrüfer's observation that to avoid pathology, an evaluation function must, among other things, have a negligible probability of underestimating a position from the perspective of the player to move.

All of the above studies have two things in common: (a) they accept the empirical evidence that the minimax principle works in practical game-playing programs and (b) they try to model mathematically the minimax algorithm and theoretically deduce what happens when heuristic values assigned to leaves are backed-up towards the root of the game tree. To make such mathematical analysis feasible, the researchers are forced to make certain assumptions about the game they model and to make simplifications in their model. Thus, the results of these models are always to be viewed with the acknowledgement of these assumptions and simplifications. In contrast to that, our approach in this article is to take (part of) a real game with a real evaluation function and observe empirically what is going on when we change the search depth and the quality of the evaluation function. We have at our disposal an absolutely correct evaluation function, which we can corrupt in a controlled way. We also have a minimax analysis engine that operates in a way similar to tablebase construction techniques to compute efficiently the results of very deep searches in our test domain.

The first part of the article, starting with Section 2, describes our choice of the game (King and Rook vs. King chess endgame, KRK for short), the evaluation function

and its artificial corruption, as well as the search engine. Section 3 presents the results for various settings of our simulation parameters and gives our explanations for the observed phenomena. In the second part of the article, consisting of Sections 4 and 5, we study the phenomenon of *bias* in minimax, noticed in KRK, more generally through experiments with randomly generated game trees. In Section 6, we provide conclusions and some ideas for further work.

2 Experimental design

We chose to centre our simulations on a simple subset of chess: the KRK endgame. In this endgame, White has a King and a Rook, while Black has only a King. The goal for White is to mate the opponent, striving to do so in as few moves as possible. There are two possible outcomes of this endgame: a win for White or a draw. While the KRK endgame is quite simple, it still possesses all the interesting attributes: positions are of various difficulty (measured in the number of moves to mate), there surely exist dependencies between the values of sibling nodes in a game tree, and there is a possibility of blunders and early termination for both sides (stalemate or losing a Rook for White; premature mate for Black). We are interested in the quality of play for White under different conditions. Therefore, unless stated otherwise, we always look at things from the White player's perspective. In addition, White is our MIN player and Black is our MAX player.

For the KRK endgame, we have at our disposal an absolutely correct evaluation function. It tells us how many moves are needed to reach mate in the case that both players play optimally, and is measured in moves. It is in the form of a database that consists of all possible legal positions and their evaluations. The database can be obtained from the UCI Machine Learning Repository [7]. The positions in the database always assume that it is Black's turn to move. There are two special cases: value 0 means Black is mated and value 255 means that Black has a draw (either the position is a stalemate or Black can capture the white Rook).

It should be noted that length-to-mate as an evaluation function requires slight modification when used with minimax. The reason is that length-to-mate decreases along the best-play line, whereas minimax backed-up values preserve the value along the best-play line. To account for this peculiarity of length-to-mate, in the experiments we used the rule $1+\mathrm{MAX}(v_i)$ instead of $\mathrm{MAX}(v_i)$. As in our experiments all the lines in the game-tree search were of equal length, this modification merely amounts to adding a constant to all the position values at the same level.

It may be argued that distance-to-mate is a rather artificial evaluation function from the usual point of view in chess where the task is to win, and not necessarily to win in the quickest way. Therefore, in chess, an evaluation function is usually interpreted as an indicator of the probability to win. However, distance-to-mate can in fact also be interpreted as such an indicator. Although virtually all the positions in KRK are won for White, an imperfect player may have difficulties in actually mating in 50 moves (the number of moves to mate allowed by the rules of chess). Such an imperfect player will have much better chances to win in a position where mate is

possible in two moves, than in a position that requires 16 moves to mate. Realistic evaluation functions for chess-playing programs also typically prefer shorter paths to mate (win) and should thus also have some correlation with our distance-to-mate.

The database consists of 28,056 positions. There are actually over 200,000 legal KRK positions, however board symmetries allow for such a reduction. Detailed descriptions of the database and board symmetries are given in Bain [3]. Our version of the database is implemented as an array of 28,056 cells and can be viewed as a sort of transposition table. Apart from positions having special evaluations of 0 or 255, there are 25,233 positions divided into 16 levels of difficulty. Positions from level 1 require one move (2 plies) to mate assuming optimal play; positions from level 2 require two moves to mate, and so on. Positions from the most difficult level require 16 moves (32 plies) to mate. Different levels have different numbers of positions; for example, there are 4,553 positions of level 14 and only 390 positions of level 16. Figure 1 shows how many cases (positions) are left unsolved if we apply searches of various depths without any knowledge apart from the rules of the game. The term 'unsolved' in this context means that White has to make a move without knowing at that time the complete move tree that guarantees a mate. The curve starts to fall significantly between depths of 14 and 20 plies and after ply 20 it steeply drops towards zero.

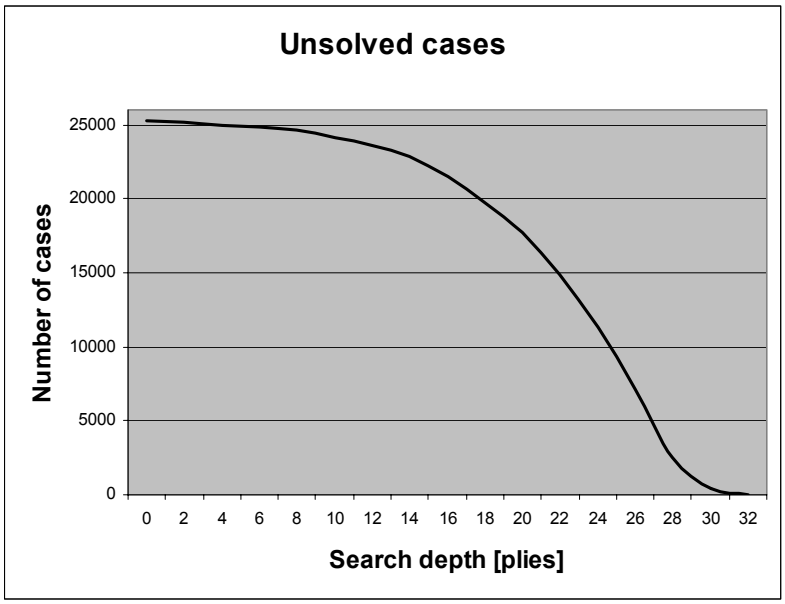
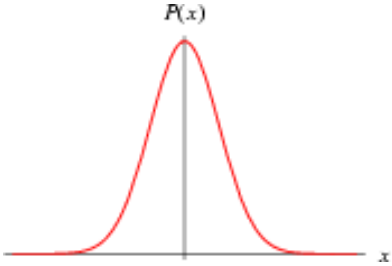


Figure 1. Number of unsolved cases as a function of search depth.

For the purpose of our experiments, we corrupted the ideal evaluation function in a controlled manner. Our method of doing this is as follows. We take a position value and add to it a certain amount of Gaussian noise. The formula and a plot are as follows:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\cdot\sigma^2)}$$



The formula gives the probability *P(x)dx* that, given the correct evaluation μ, the standard deviation σ, and the random errors, x will take on a value in the range [x, x + dx], which is a real number. The error of new evaluation is $\mu - x$. We do this for all positions in the database, including the positions where Black is already mated (special value 0). The corruption is symmetrical, meaning that there is practically equal chance that the new evaluation will be optimistic or pessimistic. We allow x to take on a negative value – in this way we are able to preserve symmetry for positions that have true values close or equal to 0.

The level of corruption is controlled by the parameter σ, the standard deviation, which controls the dispersion of the corrupted values x around the correct values μ (the width of the hill on the plot above). The standard deviation is measured in moves. For example, if σ equals 0.5, this means that approximately two thirds of the corrupted evaluations are within 0.5 moves around the true evaluation and over 95 per cent of the corrupted evaluations are within 1.0 move (two standard deviations) around the true evaluation.

To be able to compare the quality of the initial knowledge (as measured by the evaluation function) to the quality of knowledge after backing up the values with the minimax algorithm, we have to calculate the evaluation error after minimaxing. This is easy, because our search algorithm returns the backed-up values from a fixed search depth for every unique KRK position in an array exactly the same as our initial database. This array is in fact our 'backed-up' evaluation function. We thus have one such array for every search depth from 0, the initial database, to 32 ply, our chosen final search depth. We chose to measure the evaluation error by the usual root of the mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_i)^2}$$

where x_i is the backed-up corrupted value and μ_i is the true value for position i. N is the number of positions in the array. This gives us a tool to monitor directly how minimax affects the quality of the evaluation function.

Our search engine is the standard fixed-depth minimax search. The only built-in knowledge it has is the ability to detect fatal errors for White (stalemates and losing the Rook). The true value for stalemate or for losing the Rook would correspond to a distance-to-mate larger than 50 because of the draw by the 50 move rule. Such a high value is virtually impossible to corrupt with noise to the extent that it becomes

comparable to other values. Therefore, the search engine was assumed to detect in a reliable way stalemates and losses of the Rook. At the other extreme, the ability to detect mate is not given. If mate is encountered during the search, it is evaluated using the corrupted evaluation function just like every other position. This evaluation does not necessarily match the game-theoretic value for mate, which is zero. We can easily search to high search depths such as 32 plies and beyond (if desired) by exploiting the fact that the KRK endgame only has a comparatively small number of unique (under symmetries) positions (28,056) which we can store in a sort of transposition table. We start at depth 0 by loading the values from a (corrupted) database, then move on to depth 2, perform a 2-ply minimax search and use the results of the previous depth as evaluations of the leaves, store the results of a depth-2 search, move on to depth 4 and so on. This back-up procedure has a linear time complexity, contrasted to the exponential complexity of forward search.

3 Results of experiments in KRK

Below we discuss our experiments in the KRK endgame. We start with analyzing the evaluation pathology (3.1). Then we discuss the evaluation bias (3.2), followed by the decision accuracy (3.3).

3.1 Evaluation pathology

Figure 2 shows what happens to the quality of the evaluation function when we change the search depth. The x-axis represents the search depth measured in plies and the y-axis represents the evaluation error *RMSE* measured in moves. Each curve in the graph represents a different evaluation function – they differ in the level of their corruption (the initial σ). The legend marks these different evaluation functions by the size of their initial σ. The last evaluation function with initial σ of 20 is off the scale and its corruption level never drops. We performed experiments with several evaluation functions having the same initial σ, because the introduction of noise is a random process. We found out that the main characteristics remain the same for all evaluation functions with the same σ, and therefore plotted just one evaluation function with certain initial σ in all the figures.

It seems that we have to divide the evaluation functions into two groups: in the first group, we have evaluation functions with a (relatively) low initial error of less than 3.0 and in the second group those with a high initial error. The first group is a more realistic model of 'real-life' evaluation functions, while the second group contains evaluation functions with little knowledge. We can observe that the error levels of evaluation functions from the first group drop slightly with the search depth, or remain on the same level of corruption at best; some even increase. In contrast, evaluation functions from the second group only increase (practically monotonically) with an increased search depth. In no case, the error tends to decrease consistently. Some curves tend to cross one another at very high search depths; the extreme case being the 2.5 and 3.0 curves. This is due to chance. Deep searches are susceptible to high statistical variability because they quite frequently end in mate (see Figure 1). There are only 27 unique mate positions, of which some are more likely than others.

The evaluation error therefore much depends on the random error at a few of these key terminal positions.

The experiment demonstrates that for the evaluation functions tested, searching more deeply does not improve the error (*RMSE*) of the evaluation function for playing the KRK endgame, not even for those functions with a small initial level of corruption. So the known phenomenon of "evaluation pathology" is certainly present in this domain. We can try to explain this pathology by domain properties that are considered relevant: independence of position values among sibling positions [5,8], and possibility of blunders [15,1]. However, the KRK endgame undoubtedly contains dependencies between the values of sibling nodes in a game tree (we will later investigate the degree of dependency more thoroughly). The endgame is also full of possibilities for blunders on the part of the white player. White can, after all, lose the Rook in at most two moves if Black plays normally. This means that two known reasons why minimax is believed to be effective in practice are present and yet the evaluation pathology is still present as well. How can this be explained? In the sequel, we find a third reason – evaluation bias. Before we discuss this reason, we attempt to quantify the dependency between the values of sibling positions.

Beal [5] analyzed the degree of dependency between sibling nodes in terms of his clustering factor *f*. By his definition, if *all* the sibling nodes had the same value then they were considered "clustered". Factor *f* for a domain was defined as the proportion of clustered nodes in the domain. He computed *f* for the King-Pawn-King (KPK) endgame and found that the degree of clustering exceeded the critical threshold from his theoretical analysis for evaluation pathology to disappear. This indicated that evaluation pathology in KPK should not occur. We cannot apply Beal's notion of clustering to quantify analogously the sibling dependency in our KRK domain, because Beal's model operated with just two possible values: a win and a loss. In our case, position values are numerical which prevents a direct comparison with Beal's analysis. Nevertheless, we here define a degree of clustering for numerical values. Our definition is based on a comparison between the value of variance over the whole domain, and the variance among siblings only. So the definition below of our clustering factor *f* is the ratio between standard deviation with regards to siblings σ_s and the standard deviation σ over the whole domain:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(v_i - \bar{v})^2}, \qquad \bar{v} = \frac{1}{N}\sum_{i=1}^{N} v_i$$

$$\sigma_s = \sqrt{\frac{1}{\sum_{i=1}^{N} b_i}\sum_{i=1}^{N}\sum_{j=1}^{b_i}(v_{ij} - \bar{v}_{ij})^2}, \qquad \bar{v}_{ij} = \frac{1}{b_i}\sum_{j=1}^{b_i} v_{ij}$$

$$f = \frac{\sigma_s}{\sigma}$$

where v_i is the true value of the node i, v_{ij} is the true value of the j-th successor of the node i, N is the number of all positions, and b_i is the branching factor at node i.

Factor f is between 0 and 1. A lower f means stronger dependency among siblings; that is a greater degree of clustering. For our KRK domain, $f = 0.4663$. For a random KRK-like domain (with the same position-move structure, but with values as random as permitted by the minimax rule), $f = 0.9377$. This indicates a significant level of dependency between the values of sibling nodes in the KRK domain. However, as has been noted, these values of f cannot be directly compared to the clustering factor as defined by Beal [5].

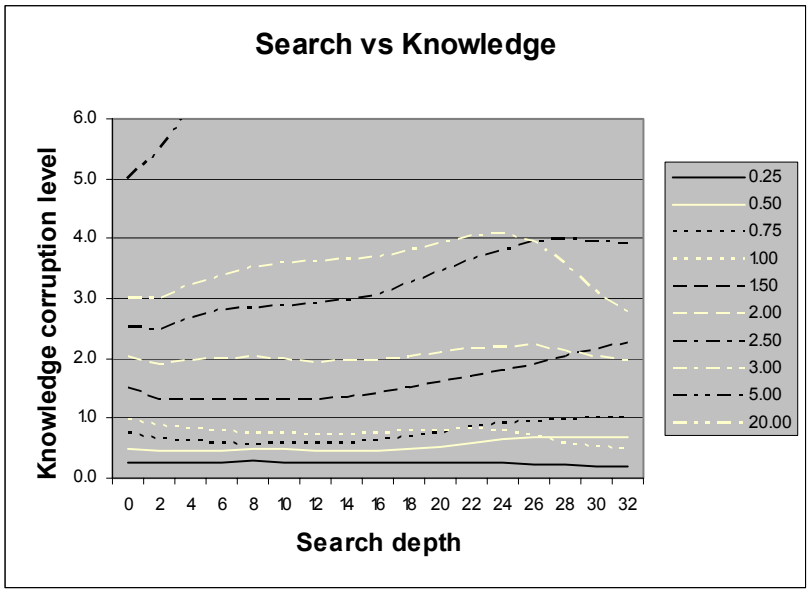


Figure 2. Influence of search depth on quality of evaluation function.

3.2 Evaluation bias

In the course of our research we were interested in *how* the backed-up evaluations were corrupted. We were curious whether the backed-up evaluations are excessively optimistic or excessively pessimistic. To this end, we calculated the bias of a backed-up evaluation function. We defined bias as:

$$bias = \frac{1}{N}\sum_{i=1}^{N}(\mu_i - x_i)$$

where μ_i and x_i are again the true and backed-up value of position i, respectively. If bias is highly negative then the backed-up values are generally overly pessimistic, and if bias is highly positive then the backed-up values are generally overly optimistic. Since the noise introduced into the various evaluation functions was symmetrical, we expected the bias to be close to zero, meaning some backed-up evaluations are too optimistic and others too pessimistic. Figure 3 charts how biased various evaluation functions are with respect to the search depth. All curves start in close proximity of zero and then without exception they all exhibit a highly positive bias. The higher the level of initial corruption the higher the bias. Most of the bias is acquired in transition from search depth 0 to search depth 2. These two levels differ the most among all of two consecutive levels – depth 0 means the algorithm goes

directly to the lookup table, while on level 2 it performs minimaxing for the first time. Other transitions just increase the depth of minimaxing.

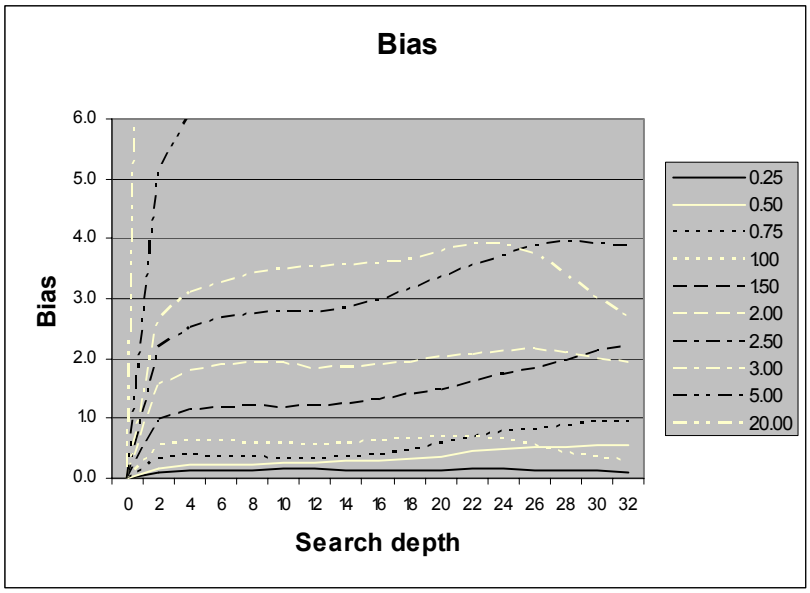


Figure 3. Influence of search depth on bias of the evaluation function.

If we look closely at what happens at the last level of minimaxing we may come up with a hypothesis why the bias occurs. On the last level, we have either a max or a min operation. In our case, we always had White to choose on the last level, which meant a min operation. In presence of noise, the operation of choosing a minimum value will be biased towards lower values. This is not to say that the value chosen will always be lower than it would have been without noise, but in general, this will be so much more often than not. We can thus see that the lowest operation of the minimax algorithm introduces a bias. However, it would appear that the opposite operation, finding the maximum, which follows on the next higher level, could compensate for this bias, at least partly. It is not obvious whether and when such compensation actually happens. We will investigate this type of compensation more thoroughly in Section 5.

If we look at Figures 2 and 3, we find out that the corruption level and the level of bias are highly correlated. For evaluation functions with a lower initial level of corruption (0.25 and 0.50) this correlation begins to manifest itself with the higher search depths of 16 to 20, while for others it begins much sooner, from a search depth of 10 for curve 0.75 and from a search depth of 4 for curve 1.0. For curves with initial corruption higher than 1.0 the correlation is strong and starts immediately from search depth 2 onward. It is no wonder that backed-up evaluation functions could not get any better – they were prevented from doing so by the bias. However, bias, at least on average, equally affects all evaluations. Its effect is similar to adding a constant to all evaluations. This in turn means that we do not change the preference order of the available moves relative to one another in the position we are trying to evaluate. If this action is correct, then minimax actually should improve the decisions with respect to the choice of moves, in spite of the evaluation pathology.

To investigate the idea of a possible improvement, we calculated another statistic, similar to *RMSE*, but adjusted to account for bias. Therefore we call this statistic "adjusted root mean squared error", or *ARMSE* for short. *ARMSE* takes into account that the values were shifted away from the true values by the bias.

$$ARMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \left(\mu_i + bias\right)\right)^2}$$

where μ_i and x_i are again the true and backed-up values of position i, respectively. How *ARMSE* changes in relation with the search depth is shown in Figure 4. Here we can see that it drastically falls with the increase of search depth. Again, the evaluation functions from the two groups defined earlier behave differently. Error levels from group one drop and then remain more or less on the same level, while error levels from group two first drop and then start to rise again. This positive effect of minimax with increasing search depth on the evaluation functions from group one is exactly the kind of result that Beal [4] was expecting from his model in 1980. However, his model with binary evaluations only does not introduce bias, and therefore cannot lead to a similar explanation of the benefit of minimax.

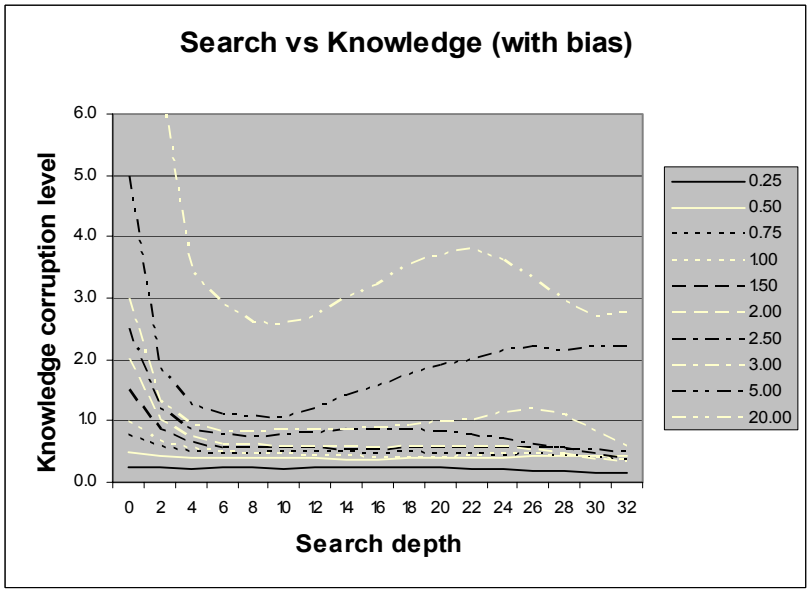


Figure 4. Influence of search depth on quality of evaluation function with bias accounted for.

3.3 Decision accuracy

Up to this point, we did not say anything about how well a computer program using one of our corrupted evaluation functions would actually play. The answer is given in Figure 5. We have played out all unique KRK positions except the ones with special values of 0 or 255, in total 25,233 positions. White was guided by a corrupted evaluation function. Additionally, White was allowed to use a simple mechanism to avoid repeating the same position over and over again. The mechanism kept a list of all positions that already occurred in the game and if the position was to be repeated

a different move was selected (the next best move according to the evaluation function). Black was always playing optimally. We measured the quality of play as the average number of moves above what an optimal white player (using a non-corrupted database) would need. This statistic is computed as White's performance loss, that is, the difference between the number of moves spent by White for all positions and the number of moves needed for all positions using optimal play, divided by the number of positions (25,233). The curves representing play using evaluation functions with initial corruption level of 5 and 20 are off the scale and result in play that is not even able to mate the opponent within the required 50 moves.

In Figure 5, we can see that an evaluation function with initial corruption level of 0.25 moves produces practically optimal play starting already at search depth 0. The quality of play using other evaluation functions gradually increases with deeper searches until it reaches a sort of threshold for a given evaluation function. From that point onward, the quality of play remains more or less on the same level. This is true for evaluation functions with initial corruption level below 3. Evaluation functions with initial corruption level higher than 3, result in a play that is not even good enough to mate the opponent within the required 50 moves. Consequently, we observe a correlation between the quality of play in Figure 5 and the evaluation error level of the evaluation functions in Figure 4.

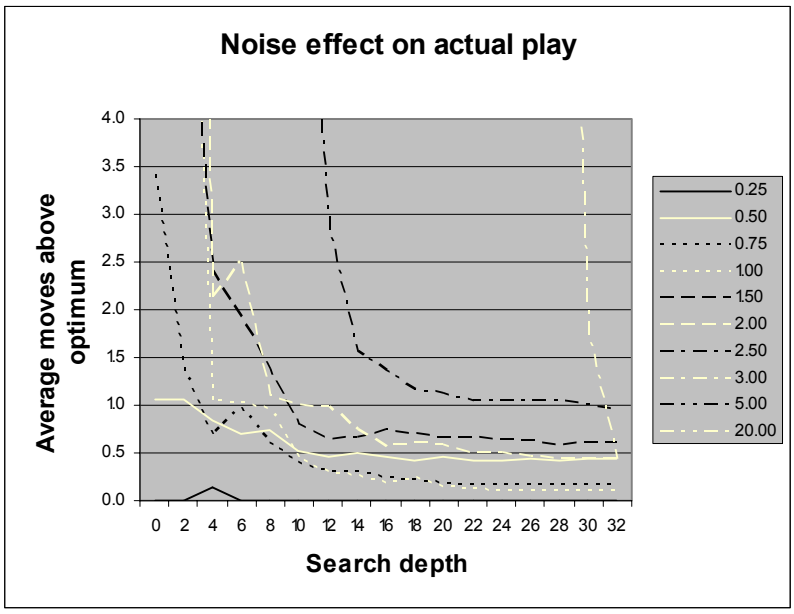


Figure 5. Quality of play using corrupted evaluation functions.

4 Minimax Error Bias in General

The notion error bias proved to be a key feature in our KRK experiments. Therefore, we decided to investigate its behaviour more generally. Since Figure 3 indicated that most of the bias is acquired at the transition from search depth 0 to search depth 2, we first studied the lowest two levels (see Figure 6).

On the lowest level, we have a set of leaf nodes. They are the only nodes of the game tree that are evaluated "statically" by an evaluation function rather than by minimax. The level just above the leaves is also quite special, since on this level minimax is applied for the very first time. It is like threading through a freshly fallen snow; bias introduced here seems to be much more significant than when we go through it for the second, or the third, or the n-th time.

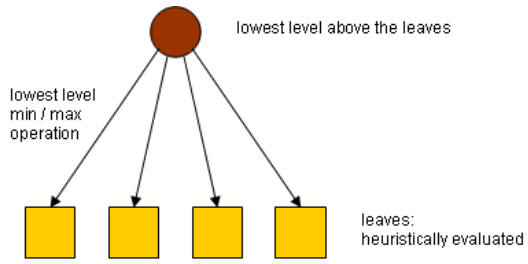


Figure 6. The lowest two levels of a game tree.

The error is introduced by the use of a heuristic evaluation function to evaluate the leaves. If we take the view of one of the two players, one of the following three possibilities may occur when evaluating the leaves:

(a) the leaf is evaluated correctly;
(b) the leaf is evaluated optimistically;
(c) the leaf is evaluated pessimistically.

Optimistic evaluation means that the leaf is evaluated as overly favourable for the player whose view we have taken. In other words, the evaluation is biased in favour of this player. The opposite is true for the pessimistic evaluation. Without loss of generality, in this article we always take the view of the player whose turn it is to move on the lowest level of the game tree above the leaves.

Our main interest was to obtain a better understanding of how bias is acquired, accumulated, and propagated through the game tree. To answer this question we performed a number of simulations and measured two statistics: *optimistic percentage* and *bias level*.

We define **optimistic percentage** as the percentage of cases where the evaluation error is biased in favour of the player whose turn it is to move on the lowest level above the leaves of the tree. For example, if we run 100 simulations and find out that in 69 of them the evaluation error is biased as stated, then the optimistic percentage is 69 per cent. One may look at the optimistic percentage statistic as a measure of the direction of bias.
Bias level is calculated as the average bias over all simulation runs. Bias level is a measure that assesses the strength of the evaluation error bias. The absolute value of the bias level tells us the strength of the bias; the higher the value, the stronger the bias. Hence, one may look at the bias level statistic as a measure of the magnitude of bias.

We started our investigation of bias by simulating the events that happen on the lowest two levels of game trees. To this effect, we generated a large number of random game trees just like the one in Figure 6. The leaf nodes were given a true evaluation, which was a random number from the interval, whose boundaries (or width) were controlled by the parameter *true value interval* (TVI). They were also given heuristic evaluations, which were obtained by adding a certain amount of noise to the true evaluations. The amount of noise was controlled by the parameter *noise level*. Noise was introduced in exactly the same fashion as in our earlier KRK experiments. We again used the Gaussian distribution and *noise level* was defined as the standard deviation from true values. Finally, the number of leaf nodes was controlled by the parameter *branching factor*. Once the leaf nodes were generated, we performed a min or a max operation and thus determined the value of the parent node. The value of the parent node is equal to the value of the best leaf node (best is either min or max, results are analogous in both cases). Bias of the parent node is the difference between its true and heuristic backed-up value. Both, optimistic percentage and bias level were measured as the average over all generated game trees. This gives us a tool that for any given setting of the control parameters enables us to estimate the direction and magnitude of the bias.

We studied the effect of changing the control parameters as follows: we varied the value of the parameter under observation while the other parameters defaulted to their predefined values. Default values for the control parameters were:

- branching factor: 20;
- interval from which the true values can be drawn: [0.0, 20.0];
- noise level: 1.50;
- distribution of heuristic errors: Gaussian.

For each experiment, a certain number of simulations were run per given value of the parameter under observation. Typically, the number of simulations was 100,000 or more. The results of the experiments are given in the following paragraphs.

In Figure 7, we see what happens to the optimistic percentage and to bias level if we vary the branching factor, starting with one and increasing it steadily to one hundred. The optimistic percentage increases monotonically with the branching factor and approaches 100 per cent. The bias level also monotonically increases (absolute value) with the branching factor, but eventually reaches its plateau and does not change much further. This can be explained quite easily. If there is only one leaf node, the symmetry of the added noise ensures that optimistic percentage is 50 per cent. If we have more than one leaf, then the following can happen. Let us look at the leaf having the best true value. Because of the symmetry of noise added, there is a 50 per cent chance that it will be optimistically biased. If it is not, however, then there is a chance that another, optimistically evaluated leaf will have a better heuristic value than the true best node. The more leaf nodes there are, the higher this chance. Therefore, the optimistic percentage increases with the increasing branching factor.

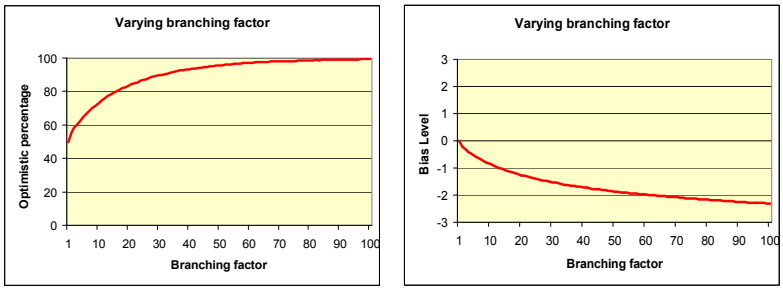


Figure 7. The effect of branching factor.

Figure 8 deals with varying the noise level, starting with zero noise. The situation is similar to the previous scenario. The optimistic percentage increases steadily with the increasing level of noise and again approaches 100 per cent. The bias level also changes monotonically, but in this scenario does not have a plateau at which it would taper off. If we again consider the true best node scenario from the previous paragraph, we see that the chance that a leaf node, which is not truly the best, will have a heuristic value better than the true best node increases with the amount of the noise being added. More noise means that nodes are more likely to change their relative order according to their values.

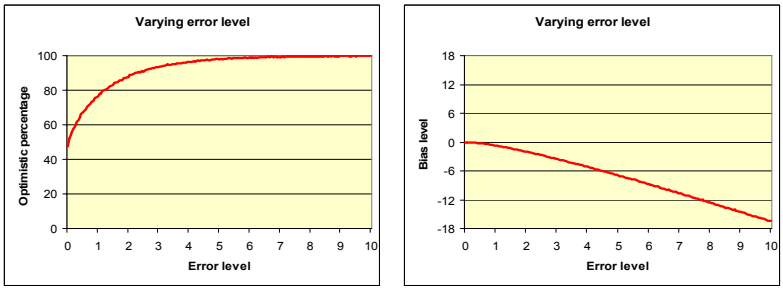


Figure 8. The effect of severity of errors.

Figure 9 explains what happens if we vary the true value interval. We kept the lower boundary of the interval constant at zero and gradually increased the upper bound from 1 to 100. The optimistic percentage, which at the start is approximately 100 per cent, monotonically drops with increasing the interval width. The bias level (observed absolutely) also reduces with increasing interval width. This is due to the change in ratio between the magnitude of true values and the level of noise.

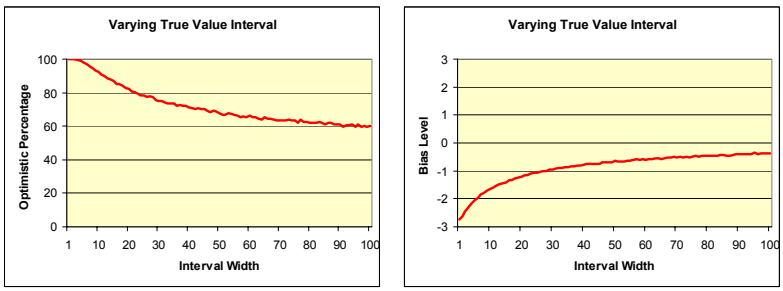


Figure 9. The effect of true value interval width.

From the experiments conducted we may conclude that the lowest min or max operation in the game tree, combined with the heuristic error made in the evaluation of the leaf nodes, produces bias that is passed on to the next upper level of the tree. The level of this bias depends on the parameters discussed in this section.

5 Multi-level Behaviour of Error Bias

Below we look at the behaviour of the error bias when the tree search goes beyond two layers. An example of such a tree (with five layers) is shown in Figure 10. The rectangle depicts a subtree – this sort of subtree was the subject of our interest thus far.

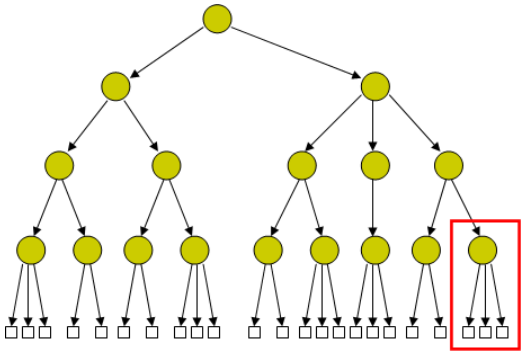


Figure 10. Extension to more than the lowest two levels of the game tree.

When simulating multiple levels we may consider another parameter, the second branching factor. Below we simulate the behaviour of two players. Each of them can have a different average number of available moves, so two branching factors may appear. We call them the odd level branching factor and the even level branching factor. At first glance it may seem unrealistic that these two differ much, however, there are many situations in practice where this is the case. KRK is one of them. White on average has about 19 moves to choose from, while Black has only 5. Apart from endgames and positions where one side has less material than the other, there are also positions where one side has a significant space advantage and consequently significantly better mobility. The effect of mobility on the minimax principle was already discussed in [6] and [9]. Below we present some further findings.

When simulating game trees with more levels both branching factors (or in fact their ratio) become the focal point of the investigation. The noise level, the true value interval, and the distribution of noise are parameters that apply only to the leaf nodes. They are properties of the evaluation function. At the same time, both branching factors are structural properties of the search tree.

Simulations presented in this section were designed as follows. We generated true and heuristic values for the leaf nodes as performed in the previous section. Then we applied the minimax procedure to these values. Finally, at the root of the tree the

backed-up values were compared as before to obtain our statistics. We generated at least 1000 game trees for each depth; the statistics were averaged.

Figure 11 shows what happens in a situation where both branching factors are (approximately) equal. The charts plot optimistic percentage and bias level against the depth of minimaxing. Level 0 is the level of leaf nodes. At this level, given that the noise is distributed symmetrically, the optimistic percentage is about 50 per cent and bias level is close to zero. Level 1 corresponds to scenarios from the previous section. We see that the bias is strong on this level. However, it is interesting that subsequent minimaxing almost immediately dissolves all the bias accumulated on that first level.

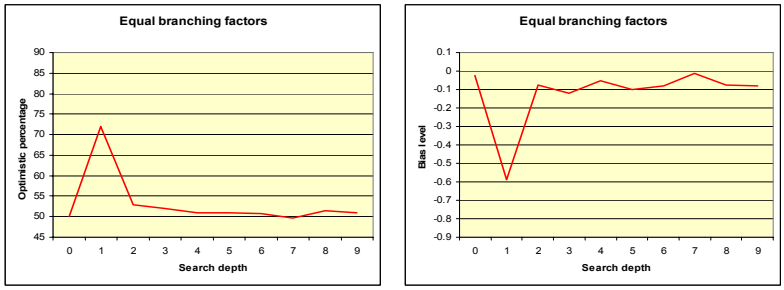


Figure 11. Approximately equal branching factors.

The next pair of charts in Figure 12 show what happens if the branching factors are not approximately equal. The ratio between them on the charts is 3 to 1 in favour of the odd branching factor. One can see that the situation is drastically different from the situation where the branching factors are about equal. The optimistic percentage rises with an increasing depth of minimaxing eventually reaching its plateau. If the ratio is even higher than 3 to 1, then the optimistic percentage approaches 100 per cent. The bias level is even more interesting. It practically remains on the level reached at the first step of minimaxing (depth 1). The oscillations are also interesting. This phenomenon has long been observed, investigated, and known in computer-chess research. The charts in Figure 12 confirm that the bias can really be the culprit for oscillations in position evaluations between odd and even depths as was suspected in [6].

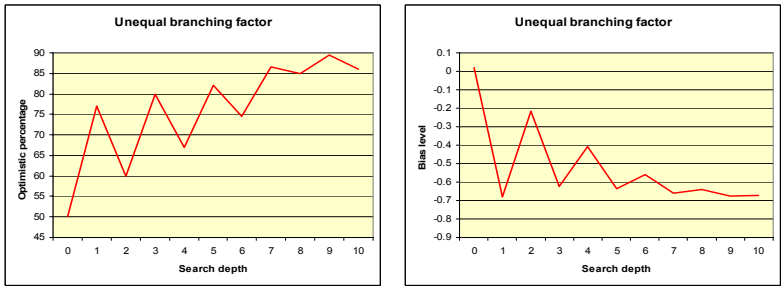


Figure 12. Unequal branching factors.

If we look at the odd levels only (cf. Figure 12), we can observe that they behave similarly to the curves in Figure 3 for the KRK experiments. In Figure 3, only every other level is plotted. Therefore, no oscillations are visible. Note that the ratio

between the branching factors in the KRK endgame is approximately 19 to 5. Somewhat analogous to this finding regarding unequal branching factors is the study on the effects of mobility by Levene and Fenner [9,10].

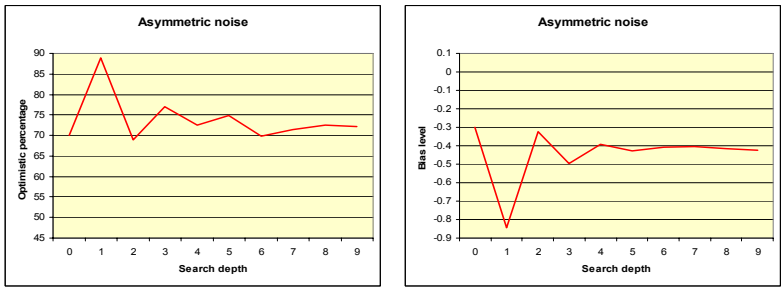


Figure 13. Asymmetrical noise and about equal branching factors.

A further point of interest is demonstrated in Figure 13. Here we have odd and even branching factors that are equal. However, the noise is not symmetrical. Noise still follows a Gaussian distribution as before, but for the purpose of our investigation, we made it asymmetrical. In the case presented in Figure 13, the chance of noise corrupting the true value optimistically is 70 per cent instead of the symmetrical 50 per cent. This is clearly visible, because at depth 0 the optimistic percentage is about 70 per cent. It is interesting that after the usual peak at depth 1, subsequent minimaxing again dissolves the bias. However, the bias is returned more or less to its initial point (here, 70 per cent optimistic and -0.3 in magnitude) and not to the symmetrical zero point as might be expected. This means that if the heuristic evaluation function has a bias for one side or the other, this bias will be preserved by minimax.

6 Conclusions and Further Work

Some theoretical studies of the minimax principle in the past surprisingly indicated that minimax might have a negative effect on the quality of position evaluation, as well as the quality of play. This finding became known as the minimax pathology. As possible answers why minimax is nevertheless successful in practice, these studies suggested two reasons: (a) dependencies between the true values of sibling nodes in a game tree, and (b) existence of traps that cause early terminations of the game.

In this article, we investigated empirically these hypotheses in a true game – we used the KRK chess endgame. The experiments indicated an evaluation pathology of minimax in this domain. Yet, regardless of that, minimax still proved successful in actual play (Figure 5), meaning that there was no decision pathology. Two known possible reasons were considered to explain the evaluation pathology. However, they did not apply convincingly in this domain: dependencies between evaluations of sibling nodes in our endgame do exist, as well as abundance of possibilities to commit blunders.

Eventually we succeeded in explaining evaluation pathology in KRK with another mechanism. We found that the minimax principle in combination with noisy evaluation functions introduces a bias into backed-up evaluations. Once such a bias is properly accounted for, the positive influence of minimax on the evaluation accuracy with an increasing search depth could be revealed. The main problem was that bias moved the backed-up evaluations away from the true values, hence causing the illusion that they were more corrupted. Yet, since it more or less affected all the evaluations equally, it did not affect the relative ordering of the available moves with respect to their quality. Therefore, we may conclude the following: if we look at the evaluations in absolute terms, they are increasingly corrupted with a growing bias, but if we look at them in relative terms, they become progressively better with higher search depths. We complete this conclusion as follows: the finding is that increasing the depth of minimax does not filter out evaluation noise, but it does help to restore the correct ordering of moves according to their strength. So, the superficial evaluation pathology is not accompanied by the decision pathology.

It may seem that it is decision accuracy only that we should be interested in, and that evaluation pathology is of no practical interest. However, knowing about evaluation pathology can be motivated by the following. Sometimes we are not interested which option is the best from a given set of options. Sometimes we need to know if the best option (or one likely to be among the best few) is good enough. For example, we are given a set of paths along which we can go. We are interested in how long the shortest path will take. However, we are constrained in time; it is no good to us if the best route takes more time than we have. In this case, it is not enough that we can (reliably) distinguish the shortest path. We have to obtain a good estimate on *how long* it will take; it is no longer sufficient to know that it will take *less time* than other paths.

We can draw another example from the game of chess. Sometimes, perhaps when analyzing a position, we are interested in *the value* of this position and not just what is deemed to be *the best move* in this position. For example, one might reason that several moves are playable, but that in general White has an advantage of, say, one half of a Pawn.

We also studied bias by simulations in randomly generated game trees. We observed two statistics, called optimistic percentage and bias level. These experiments show that most of the bias is accumulated at the bottom level of minimax. This bias starts disappearing at higher levels of the minimax tree, unless branching factors at odd and even levels of the tree are significantly different. So, the difference in the mobility between the two players becomes critically important.

It would be interesting to reproduce our experimental findings using a more complex real endgame, especially one where the ratio between White's and Black's branching factors is closer to 1. The chess endgame KQKBN comes to mind. A further study of how the bias behaves and what affects it would also be of interest. Especially interesting would be to investigate what happens with the bias if we mix the

functions (min and max) at the lowest level of minimaxing – some branches we search to even depths, the others to odd depths.

Acknowledgement

We would like to thank wholeheartedly the anonymous referees for a very thorough job that prompted us to rethink and improve our work significantly.

References

[1] B. Abramson. An Explanation of and Cure for Minimax Pathology, in: L.N. Kanal and J.F. Lemmer (Eds.), Proceedings of the 1st Conference on Uncertainty in Artificial Intelligence, North-Holland Publishers, 1986, pp. 495-504.
[2] I. Althöfer. Generalized minimax algorithms are no better error correctors than minimax itself, in: D.F. Beal (Ed.), Advances in Computer Chess 5, Elsevier Science Publishers, Amsterdam, 1989, pp. 265-282.
[3] M. Bain. Learning optimal chess strategies, in: S. Muggleton (Ed.), Proc. Intl. Workshop on Inductive Logic Programming, Institute for New Generation Computer Technology, Tokyo, 1992.
[4] D.F. Beal. An analysis of minimax, in: M.R.B. Clarke (Ed.), Advances in Computer Chess 2, Edinburgh University Press, Edinburgh, 1980, pp. 103-109.
[5] D.F. Beal. Benefits of minimax search, in: M.R.B. Clarke (Ed.), Advances in Computer Chess 3, Pergamon Press, Oxford, 1982, pp. 1-15.
[6] D.F. Beal, M.C. Smith. Random Evaluations in Chess, ICCA Journal 17 (1) (1994) 3-9.
[7] C.L. Blake, C.J. Merz. UCI Repository of machine learning databases, Department of Information and Computer Science, University of California, Irvine, CA, 1998, [http://www.ics.uci.edu/~mlearn/MLRepository.html].
[8] I. Bratko, I., M. Gams. (1982). Error analysis of the minimax principle, in: M.R.B. Clarke (Ed.), Advances in Computer Chess 3, Pergamon Press, Oxford, 1982, pp. 1-15.
[9] M. Levene, T.I.Fenner. A Partial Analysis of Minimaxing Game Trees with Random Leaf Values, ICCA Journal 18 (1) (1995) 20-33.
[10] M. Levene, T.I. Fenner. The Effect of Mobility on Minimaxing of Game Trees with Random Leaf Values, Artificial Intelligence Journal, 130 (1) (2001) 1-26.
[11] D.S. Nau. Quality of Decision Versus Depth of Search on Game Trees, PhD dissertation, Duke University, 1979.
[12] D.S. Nau. Preliminary results regarding quality of play versus depth of search in game playing, in: Proceedings of the First International Symposium on Policy Analysis and Information Systems, 1979, pp. 210-217.
[13] D.S. Nau. An Investigation of the Causes of Pathology in Games, Artificial Intelligence 19 (1982) 257-278.
[14] D.S. Nau. Pathology on Game Trees Revisited, and an Alternative to Minimaxing, Artificial Intelligence 21 (1-2) (1983) 221-244.
[15] J. Pearl. Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley Publishing Company, Reading, MA, 1984.
[16] G. Schrüfer (1986) Presence and absence of pathology on game trees, in: D.F. Beal (Ed.), Advances in Computer Chess 4, Pergamon Press, Oxford, 1986, pp. 101-112.