Available online at www.sciencedirect.com
SCIENCE DIRECT®

Theoretical Computer Science 349 (2005) 245–267

Theoretical Computer Science

www.elsevier.com/locate/tcs

Selecting evaluation functions in Opponent-Model search

H.H.L.M. Donkers*, H.J. van den Herik, J.W.H.M. Uiterwijk

Department of Computer Science, Institute for Knowledge and Agent Technology, Universiteit Maastricht, P.O. Box 616, 6200 MD Maastricht, The Netherlands

Abstract

Opponent-Model search is a game-tree search method that explicitly uses knowledge of the opponent. There is some risk involved in using Opponent-Model search. For adequate forecasting, two conditions should be imposed. Both the prediction of the opponent's moves and the judgement of future positions should be of good quality. The two conditions heavily depend on the evaluation functions used.

In the article we distinguish evaluation functions by type. Three fundamentally different types are introduced. Thorough analysis of a variety of characteristics leads to eight possible orderings. The role of the evaluation functions is studied by attempting to answer five research questions. Moreover, actual computer game-playing programs investigate the research questions by a series of experiments in which Opponent-Model search is performed. The game of Bao is our test bed, it was selected because of its relatively narrow game tree, which allowed for an appropriate search depth in the experiments. We restrict ourselves to five evaluation functions generated with the help of machine-learning techniques. A set of round-robin tournaments between these evaluation functions show that when the above conditions are met, Opponent-Model search can be applied successfully. Answers to the research questions are given in the conclusions.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Opponent models; Search; Evaluation functions; Bao

1. Five research questions

Since the first scientific formulation of Opponent-Model search (OM search) in 1993 [2,12] as an alternative for the widely used α–β search procedure, several issues of OM search have been studied. We mention three of them: (1) efficient implementations and pruning [3,5]; (2) the risks involved [8]; (3) *Probabilistic* OM Search [7]. As matters stand at present, it is clear that OM search as originally defined is hardly an alternative for α–β search [5]. However, we are convinced that under suitable circumstances, OM search might be used successfully.

In our investigations we aim at creating the right circumstances under which it is possible to apply OM search with success, i.e., OM search plays better than the methods conventionally used. We are interested in particular in the role of heuristic evaluation functions in OM search, and how their selection and exploitation influences the effectiveness of OM search.

* Corresponding author. Tel.: +31 43 3883481; fax: +31 43 3884897.
E-mail addresses: donkers@cs.unimaas.nl (H.H.L.M. Donkers), herik@cs.unimaas.nl (H.J. van den Herik), uiterwijk@cs.unimaas.nl (J.W.H.M. Uiterwijk).

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.09.049

In the original formulation of OM search [2,12], it is stated that the own evaluation function should be better than the one that is used to model the opponent in OM search. Such a demand follows from our intuition. Provided that the evaluation function used for the opponent is better than our evaluation function, it seems rational to use the opponent's evaluation function in α–β search, instead of using OM search at all. Yet, there is something to ponder: so far, it is unclear how to define 'better' or how to order evaluation functions. This area is almost unexplored. To shed some light on the intricacies we formulate five research questions. They are introduced below together with the course of the article.

In Section 2 we give a brief overview of OM search. Section 3 discusses three types of evaluation functions, viz. seen as an estimate of prediction, probability, and profitability. Then we pose our first research question: *how can evaluation functions be ordered*? In Section 4 we offer a range of eight possible orderings of which some are merely of theoretical interest but others have practical use, especially the operational ordering. Defining an ordering on evaluation functions implies that they are comparable. This is not inherently true since the interpretation of evaluation functions differs among researchers and domains. After the discussion on the ordering of evaluation functions, we adhere to the operational ordering of evaluation functions for the set-up of the experiments.

In Section 5, the discussion on the risk involved when using OM-search reveals that it is important to predict the opponent's moves as good as possible. Our second research question reads: *to what extent does the quality of the prediction influence the performance of OM search*? We tested the importance by varying the search depth used for the prediction of the opponent's move. We hypothesized that a larger search depth will lead to better results, in comparison to α–β.

Section 6 describes how we generated five evaluation functions of increasing (operational) quality. We expect that the best results in comparison to α–β will be achieved by using a high-quality function for the own evaluation function against a lower-quality function for the opponent's function. So, our third research question reads: *does the relative quality of the two evaluation functions influence the performance of OM search*?

Meanwhile, the discussion on the risk involved when using OM search continues and hints at the importance of the quality of the own judgement. Therefore, our fourth research question reads: *does the quality of the own judgements influence the performance of OM search*? We tested this importance by varying the search depth used for the own judgement. We hypothesized that a larger search depth will lead to better results. Naturally, allowing OM search to spend more resources by additional search depth should be compared to the conventional α–β search with the same additional search depth. This leads to the fifth and most important research question: *can OM search with additional search depths outperform α–β search with the same additional search depths*?

To the extent possible we answered the research questions theoretically. In addition, we performed a series of tournaments with the game of Bao (explained in Appendices A and B). Section 7 provides the experimental setup. The experiments and their results are discussed in Section 8. Our five research questions are answered in Section 9 by providing final conclusions.

2. Opponent-Model search

OM search [2,3,5,8,12] is a game-tree search algorithm that uses a player's hypothesized model of the opponent in order to exploit weak points in the opponent's search strategy. The original formulation of the OM-search algorithm is based on three strong assumptions concerning the opponent and the player:

(1) the opponent (called MIN) uses minimax (or an equivalent algorithm) with an evaluation function (V_{op}), a search depth and a move ordering that are known to the first player (called MAX);
(2) MAX uses an evaluation function (V_0) that is *better* than MIN's evaluation function;
(3) MAX searches at least as deep as MIN.

In a nutshell, the OM-search procedure prescribes that MAX maximizes at max nodes, and selects at min nodes the moves that MAX believes MIN would select. Below, we provide a brief technical description of OM search, its notation, the relations between the nodes in the search tree, and some suggestions for an efficient implementation. For an extensive description of OM search we refer to [5].

OM search can be described by the following equations, in which $V_0(\cdot)$ and $V_{\mathrm{op}}(\cdot)$ are the evaluation functions, and $v_0(\cdot)$ and $v_{\mathrm{op}}(\cdot)$ are the node values. Subscript '0' is used for MAX values, subscript '*op*' is used for MIN values.

```
OmSearch($P$)
1   if ($P$ is a leaf node) return ($V_0(P)$, null)
2   if ($P$ is a max node)
3       $j^* \leftarrow$ null ; $v_0^* \leftarrow -\infty$
4       for all (moves $j$ at $P$)
5           $(v_0, \_) \leftarrow$ OmSearch($P_j$)
6           if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $j^* \leftarrow j$
7   if ($P$ is a min node)
8       $(v_{op}^*, j^*) \leftarrow$ AlphaBetaSearch($P, -\infty, \infty, V_{op}(\cdot)$)
9       $(v_0^*, \_) \leftarrow$ OmSearch($P_{j^*}$)
10  return ($v_0^*, j^*$)
```

Fig. 1. OM search with α–β probing. We use j to indicate moves and P_j to indicate the child node of P that is reached when playing move j.

(The notation $\min \arg_i f(i)$ denotes the value of i for which $f(i)$ is minimal.)

$$v_0(P) = \begin{cases} \max_j \ v_0(P_j) & \text{max nodes,} \\ v_0(P_j), \quad j = \min \arg_i v_{\text{op}}(P_i) & \text{min nodes,} \\ V_0(P) & \text{leaf nodes.} \end{cases} \tag{1}$$

$$v_{\text{op}}(P) = \begin{cases} \max_j \ v_{\text{op}}(P_j) & \text{max nodes,} \\ \min_j v_{\text{op}}(P_j) & \text{min nodes,} \\ V_{\text{op}}(P) & \text{leaf nodes.} \end{cases} \tag{2}$$

If P is a min node at a depth larger than the search-tree depth of the opponent, then $v_0(P) = \min_j \ v_0(P_j)$.

2.1. Implementation

For a search tree with branching factor w and even fixed depth d, OM search needs exactly $n = w^{d/2}$ evaluations of $V_0(\cdot)$ to determine the root value, since the search strategy is as follows: in each max node *all* w children are investigated and in each min node, *only one* child is investigated (see [7]). Because the OM-search value is defined as the maximum over all these n values of $v_0(\cdot)$, none of these values can be missed. This means that the efficiency of OM search depends on how efficient the values for $v_{\text{op}}(\cdot)$ can be obtained.

A straightforward and efficient way to implement OM search is by applying *α–β probing*: at a min node it starts performing α–β search with the opponent's evaluation function (the *probe*), and thereafter it performs OM search with the move that α–β search has returned; at a max node, it maximizes over all child nodes. The probes can be efficiently implemented by using an enhanced form of α–β search. Because for every min node, a separate probe is performed, many nodes are visited during multiple probes. (For example, every min node P_j on the principal variation of a node P will be visited at least twice.) Therefore, the use of transposition tables leads to a major reduction of the total number of nodes visited. The basic version of the algorithm is given in Fig. 1.

The α–β probes at a min node P and at each grandchild (min nodes P_{j_k}) are not independent since the α–β value of P, $v_{\text{op}}(P)$, is necessarily equal to or larger than all α–β values of P_{j_k}. This means that $v_{\text{op}}(P)$ can be used to reduce the window of the probes at the grandchild nodes by setting the β parameter of the probe at P_{j_k} to $v_{\text{op}}(P) + 1$. For a further analysis of pruning in OM search, we refer to [5,8].

OM search assumes that MAX speculates on all min nodes about the move that MIN is going to choose. In deeper parts of the search tree, the prediction of MIN's move is based on shallower α–β probes than higher in the tree. It could therefore be justified to speculate only in the upper portion of the search tree. Fig. 2 gives an algorithm for OM search with restricted speculation (OmSearchRs). The algorithm is intentionally presented in a simplified version. For a discussion on more efficient implementations we refer to [5].

```
**OmSearchRs**$(P)$
1  **if** (do not speculate at $P$) **return AlphaBetaSearch**$(P, -\infty, \infty, V_0(\cdot))$
2  **if** ($P$ is a max node)
3      $j^* \leftarrow$ **null** ; $v_0^* \leftarrow -\infty$
4      **for all** (moves $j$ at $P$)
5          $(v_0, \_) \leftarrow$ **OmSearch**$(P_j)$
6          **if** $(v_0 > v_0^*)$ $v_0^* \leftarrow v_0$ ; $j^* \leftarrow j$
7  **if** ($P$ is a min node)
8      $(v_{op}^*, j^*) \leftarrow$ **AlphaBetaSearch**$(P, -\infty, \infty, V_{op}(\cdot))$
9      $(v_0^*, \_) \leftarrow$ **OmSearch**$(P_{j^*})$
10 **return** $(v_0^*, j^*)$
```

Fig. 2. OM search with restricted speculation. The only difference with the algorithm in Fig. 1 is in the first line.

3. Three types of evaluation functions

Evaluation functions play a central role in OM search since the opponent model is primarily based on an evaluation function. There is an intuitive feeling that one should not use OM search if the opponent's evaluation function is better than the own evaluation function, since in such a case it seems better to adopt the opponent's evaluation function and use (an equivalent of) Minimax search. However, the term 'better' is not easy to define. In this section we will examine three types of evaluation functions: *predictive*, *probability estimating*, and *profitability estimating* evaluation functions. In the next section we will investigate how evaluation functions can be compared to each other.

Static heuristic evaluation functions (or evaluation functions for short) in game-tree search algorithms are used to replace the game-theoretic value of game positions that have been reached during search but are not searched any further (they are leaf nodes in the search tree). For an introduction to heuristics in general and to static evaluation functions in particular, we refer to [14].

In normal Minimax search, an evaluation function returns a scalar value that measures the strength of that position for MAX. If a position happens to be a terminal position of the game (or the evaluation function can deduce the true value), then the evaluation function returns some encoding of the game-theoretic value (usually win, loss, or draw). Otherwise, the evaluation function returns a heuristic value. A common encoding scheme for evaluation functions is as follows: heuristic values lie within some range $[-H, H]$; a heuristic draw estimate is rewarded by 0. A *proven win* for MAX is rewarded by some large value $W > H$; and a *proven loss* for MAX by $-W$. The depth at which a win or loss is found (relative to the root of the search tree) is usually incorporated in the value: $W - d$ for a win at depth d and $d - W$ for a loss at depth d. The idea is that the proven win and loss values both have a value range different from the heuristic values. The proven draw value (zero) coincides with the heuristic draw value. The heuristic values can be interpreted in three different ways, leading to three types of evaluation functions.

In the first type, the heuristic value is viewed as a *predictor* for the true game-theoretic value of a position: the higher the heuristic value for a position, the more certain it is that the position is a game-theoretic win. This interpretation is used by Pearl [14] and others in their theoretical study of Minimax search and α–β search.

In the second type, the heuristic value is viewed as some encoding of the *probability* to win the game at that position. This interpretation is used by authors that use machine-learning techniques such as temporal-difference learning to achieve evaluation functions (cf. [1]). The probability to win the game from a given position depends on the specific opponent and on a number of game settings such as the start position, the search time, and the search technique used.

The third type of evaluation function views the heuristic value as a measure of *profitability* of the position for MAX. It is difficult to define the notion 'profitability of a position' formally, although in our view the concept is used in many knowledge-based, hand-made heuristic evaluation functions. (According to [14] the term 'heuristic evaluation' itself points to the notion of profitability.) Here we want to discriminate between the estimated profitability as used in a specific evaluation function and the true profitability itself.

We note that the profitability is only partly based on the prediction of the game-theoretic value, since it also incorporates the possibility of errors by both sides. In addition, it includes MAX's own strengths and weaknesses. We stress that the profitability of a position is independent of the actual opponent and of the actual game setting. For instance, a position might be a game-theoretic win, but the winning solution is difficult to achieve because of many threats and many possible errors. Then this position is clearly less profitable than an easy win. In an extreme contrast, it may happen that a position that is a game-theoretical loss could offer so many opportunities to exploit an opponent's error, that the position may become profitable. (Some human players, former Chess World Champion Emanuel Lasker among them, show(ed) a preference for complex positions in which they face a slight disadvantage.)

Analogously to the case of the win probabilities above, the profitability of positions is dependent on external factors. When an evaluation function is used that truly measures the profitability, and the anticipated circumstances are present, then a program is expected (but not guaranteed) to win. However, the profitability of positions can drop or raise drastically when the circumstances change. For instance, when a program is unexpectedly confronted with a perfectly playing opponent, the profitability of positions must become close or equal to the game-theoretic value in order to provide adequate counterplay.

Despite all advantages of including the circumstances, it is a further complication that the profitability of a position is just as difficult to determine as the game-theoretic value. A practical evaluation function can only *estimate* the profitability of a position.

4. Comparing the evaluation functions

When building an evaluation function for a specific game, there are many decisions to take. Consequently, there are many different evaluation functions possible for a game. In order to compare the evaluation functions and to select the 'best' one, a formal analysis of evaluation functions and their mutual relations is needed. Multiple orderings of evaluation functions appear to be possible: below we define eight different orderings. Not all orderings are suitable for all evaluation functions—if two evaluation functions are of different type (as described in the previous section) they are only comparable in a restricted way. Therefore, we will group the eight possible orderings according to three ordering principles: (1) ordering on *win prediction*, (2) ordering on *operational behaviour*, and (3) ordering on *profitability estimation*. The first principle is based on the predictive interpretation of evaluation functions and the third principle is based on the profitability interpretation. The ordering on operational behaviour is closest related to the probability interpretation of evaluation functions, but can be used with all three interpretations.

4.1. Ordering on win prediction

The first principle according to which evaluation functions can be ordered is on the degree in which they predict the game-theoretic values. As a consequence, this ordering principle is only suitable for predicting evaluation functions. Assume that the true payoffs in a game G are restricted to: $\{loss, draw, win\}$ (where $loss < draw < win$) and that the ranges of the heuristic evaluation functions (the true values included) are restricted to the interval $[-W, W] \subset \mathbb{Z}$.[1] Let $\mathbf{V} : L \to \{loss, draw, win\}$ denote the true game-theoretic value of the leaf positions L, and let $\mathcal{V}$ denote the set of all evaluation functions $V : L \to [-W, W]$.

Definition 1. An evaluation function V is called a *perfect predictor* if for all positions $P, Q \in L$ holds that if $\mathbf{V}(P) < \mathbf{V}(Q)$ then $V(P) < V(Q)$.

Definition 1 implies that a function $t : [-W, W] \to \{loss, draw, win\}$ must exists such that for all $P \in L$ holds that $t(V(P)) = \mathbf{V}(P)$. A perfect predictor makes any search superfluous, provided that t is known.

The definition of a perfect predictor is equivalent to the definition of perfect play [4]. In Christensen and Korf's definition, an evaluation function performs perfect play if it exhibits a perfect outcome determination and is move invariant. Outcome determination means that the evaluation function returns game-theoretic values for terminal positions and move invariance means that making optimal moves does not change the evaluation value.

[1] We disregard real-valued evaluation functions since the set L of leaf nodes is a finite set. Any evaluation function $f : L \to \mathbb{R}$ can be mapped isomorphically to a function $V : L \to \mathbb{Z}$ such that $f(P_1) \leqslant f(P_2) \Leftrightarrow V(P_1) \leqslant V(P_2)$.

Definition 2. An evaluation function $V \in \mathcal{V}$ is called a *partial predictor* if a subset $L' \subset L$ exist such that for all $P, Q \in L'$ holds that if $\mathbf{V}(P) < \mathbf{V}(Q)$ then $V(P) < V(Q)$. Any such subset L' is called a *perfectly predicted set* (PPS) of V.

This definition implies that the empty set and any singleton subset of L are PPSs. However, we are particularly interested in maximal perfectly predicted sets.

Definition 2a. A *maximal perfectly predicted set* (MPPS) L^* of V is a PPS such that for all PPSs L' of V, $|L^*| \geqslant |L'|$.

There can exist multiple MPPSs for an evaluation function V, but they will by definition all have the same size. Since L is finite, the size of an MPPS is well defined. The first ordering of evaluation functions is based on the size of the MPPSs.

Ordering 1. Evaluation function $V_1 \in \mathcal{V}$ is called a *better predictor* than function $V_2 \in \mathcal{V}$ if the MPPSs of V_1 are larger than those of V_2.

It is possible to define an ordering of evaluation functions on the basis of the following domination relation, but since this is only a *partial* ordering,[2] it will not be of major use.

Ordering 2. Evaluation function $V_1 \in \mathcal{V}$ is said to be a *dominating predictor* with respect to $V_2 \in \mathcal{V}$ if at least one MPPS of V_2 is a proper subset of an MPPS of V_1.

In the case of most games that are of interest, the maximal perfectly predicted set of positions is small and will mainly occur at the end of the game. To be able to classify and order evaluation functions in a more useful way, we could relax the mapping function t (see above). One way to relax t is to add some probability and statistics. For instance, take a sample $L_i \subseteq L$ of leaf positions and determine for every position both the evaluation value and the game-theoretic value. The sample should be representative for real game positions. Determine statistically the influence of the game-theoretic value on the evaluation value. If the game-theoretic value has no significant influence on the evaluation value, then, presumably, the evaluation function is also not a good predictor of the game-theoretic value. The exact statistical procedure depends on the shape of the samples and on the number of game-theoretic values. For instance, one could just determine the (statistical) P-value of the test on the difference between win positions and loss positions and use this P-value to order the evaluation functions. This leads to our third, informally defined, ordering of evaluation functions.

Ordering 3. Evaluation function $V_1 \in \mathcal{V}$ is called a *better statistical predictor* than function $V_2 \in \mathcal{V}$ if the above statistic procedure provides a lower P-value for V_1 than for V_2.

The three orderings above depend on the interpretation of evaluation functions as predictors for the game-theoretic value. The next ordering principle is of more practical use since it can be based on all three interpretations, including the probabilistic interpretation.

4.2. Ordering on operational behaviour

The second principle on which evaluation functions can be ordered is based on their operational behaviour. We reiterate that it is the only ordering principle that is suitable for all three types of evaluation functions. The principle leads to the fourth ordering.

Ordering 4. Evaluation function $V_1 \in \mathcal{V}$ is called *operationally better* than evaluation function $V_2 \in \mathcal{V}$ if player A using V_1 achieves a higher score in a given tournament than player B using V_2 instead of V_1. (All other circumstances are equal to A and B.)

[2] This means that not every pair of evaluation functions is comparable in this respect.

The setup of the tournament influences the exact meaning of this operational ordering of evaluation functions. Some important factors are: (1) the set of used opening positions, (2) the search depth, and (3) the size of the tournament: (3a) other players are also included or (3b) these two players are only playing against each other. The operational ordering of evaluation functions is an overall quality measurement of the evaluation functions. The operational ordering is used in the field of automatically learning evaluation functions (for instance, in genetic algorithms and temporal-difference learning [1]).

Another ordering used in automated evaluation-function learning is related to *move invariance* as explained in [4]. In this ordering, an evaluation function is preferred if it shows less difference between the static evaluation function of a position and the Minimax value of a game tree at the same position. Samuel [16,17] used this ordering implicitly in his pioneering work on the game of Checkers. In fact, the ordering is not used to compare two different evaluation functions, but to change an evaluation function in a desired direction. This learning method leads to win-predicting evaluation functions.

4.3. Ordering on profitability estimation

The third ordering principle of evaluation functions is based on the estimation of the profitability per position. As a consequence, this ordering principle is suitable for profitability estimating evaluation functions only. The principle is quite opposite to the operational one since profitability is not as easily measurable as the number of wins in a tournament. For a start, we have to assume that an evaluation function $\mathbb{V} : L \to \mathbb{Z}$ exists that measures the exact profitability of all leaf positions $P \in L$. The higher the value of $\mathbb{V}(P)$, the higher the profitability is. We start with a general definition.

Definition 3. Two evaluation functions $V_1 \in \mathcal{V}$ and $V_2 \in \mathcal{V}$ are *equivalent* on L if they order all positions equally: $\forall P_a, P_b \in L : V_1(P_a) \leqslant V_1(P_b) \iff V_2(P_a) \leqslant V_2(P_b)$.

Equivalent evaluation functions are indistinguishable for search algorithms. This leads us to the definition of perfect evaluation functions.

Definition 4. A *perfect* evaluation function $V \in \mathcal{V}$ is a function that is equivalent to $\mathbb{V}$.

Analogously to the perfectly predicting evaluation function, a perfect profitability evaluation function eliminates the need for any search. However, it is obvious that one never knows whether a profitability evaluation function is perfect. Therefore, we define that a *good* evaluation function is a good estimation of $\mathbb{V}$.

The fifth ordering of evaluation functions is analogous to the first ordering: it compares the evaluation functions on the size of the set for which they are equivalent with $\mathbb{V}$. For convenience we define a maximal profitability equivalence set (MPES).

Definition 4a. A *profitability equivalence set* (PES) L' of V is a subset of L such that V restricted to L' is equivalent to $\mathbb{V}$ restricted to L'. A *maximal profitability equivalence set* (MPES) L^* of V is a PES such that for all PESs L' of V, $|L^*| \geqslant |L'|$.

Similar to MPPSs (see Definition 1a), there can be multiple MPESs for one evaluation function, but all MPESs for one evaluation function are of the same size.

Ordering 5. Evaluation function $V_1 \in \mathcal{V}$ is called a *better profitability estimator* than evaluation function $V_2 \in \mathcal{V}$ if the MPESs of V_1 are larger than those of V_2.

In the case of a profitability estimation, a partial ordering of evaluation functions, similar to ordering 2, can be defined on the basis of a domination relation.

Ordering 6. Evaluation function $V_1 \in \mathcal{V}$ is a *dominating profitability estimator* with respect to $V_2 \in \mathcal{V}$ if at least one MPES of V_2 is a proper subset of an MPES of V_1.

Analogously to our third ordering, the seventh ordering uses the degree of correlation.

Ordering 7. Evaluation function $V_1 \in \mathcal{V}$ is called a *statistically better profitability estimator* than evaluation function $V_2 \in \mathcal{V}$ if V_1 and $\mathbb{V}$ are higher correlated than V_2 and $\mathbb{V}$.

Finally, evaluation functions can be ordered on their rank-difference sum[3] with $\mathbb{V}$. This sum is calculated by first rank-transforming the evaluation values of all positions for V and $\mathbb{V}$ and then computing $\sum_P |rank(P, V)-rank(P, \mathbb{V})|$. This produces our eighth ordering of evaluation functions.

Ordering 8. Evaluation function $V_1 \in \mathcal{V}$ is called a *better ranked profitability estimator* than evaluation function $V_2 \in \mathcal{V}$ if the rank-difference sum of V_1 with $\mathbb{V}$ is smaller than the rank-difference sum of V_2 with $\mathbb{V}$.

The advantage of this method is that equivalent evaluation functions obtain the same rank-difference sum. A disadvantage is, of course, that this sum cannot be computed in practice. The ranking approach to evaluation functions also gives rise to the following definitions of overestimation and underestimation (which we both will call estimation error).

Definition 5. An evaluation function $V \in \mathcal{V}$ *overestimates* a position P if the rank of $V(P)$ is higher than the rank of $\mathbb{V}(P)$.

Definition 6. An evaluation function $V \in \mathcal{V}$ *underestimates* a position P if the rank of $V(P)$ is lower than the rank of $\mathbb{V}(P)$.

4.4. Short discussion on ordering evaluation functions

The analysis of evaluation functions and their ordering makes it clear that a straightforward use of the relation ‘better’ with respect to evaluation functions is not possible. So, the intriguing question remains: which one of the following orderings deserves our preference?
(1) V_1 is a better predictor than V_2;
(2) V_1 is a dominating predictor with respect to V_2;
(3) V_1 is a better statistical predictor than V_2;
(4) V_1 is operationally better than V_2;
(5) V_1 is a better profitability estimator than V_2;
(6) V_1 is a dominating profitability estimator with respect to V_2;
(7) V_1 is a statistically better profitability estimator than V_2;
(8) V_1 is a better ranked profitability estimator than V_2.
The orderings will have strong correlations in the sense that if V_1 is better than V_2 in one of the orderings, it will probably also be better in another one of the orderings (except for the two domination orderings 2 and 6, because these are partial orderings). However, the orderings are not equivalent.

Some of the orderings appear only to be of theoretical use since the exact computation of those orderings is as hard as obtaining the game-theoretic value or the true probability of all positions. However, it should be possible to design statistical methods to estimate the orderings, even if they are based on win prediction or profitability. As stated above, two of the orderings (the dominating orderings 2 and 6) are partial orderings, which means that not all evaluation functions are comparable. The usage of such orderings is limited. In addition to these considerations, we remark the following: if one evaluation function dominates another one as a profitability estimator, then this might be a stronger argument to use it than that the evaluation function is, for instance, a statistically better profitability estimator. Obviously, the most directly applicable ordering is (4), the ordering based on operational behaviour.

The selection of one of the orderings should be made by a three-step procedure: (1) the type of the evaluation functions according to Section 3, (2) the purpose of the comparison of evaluation functions, and (3) the practical circumstances.

[3] This is similar to Spearman’s Rank Correlation Coefficient [18], which can also be used here.

5. Risk in Opponent-Model search

Although using knowledge of the opponent during search seems obvious and OM search seems to be a reasonable approach to do so, there are three different types of risk involved when OM search is used. If these risks are not taken seriously, OM search is bound to fail. The risks are: (1) imperfect knowledge of the opponent, (2) overestimation of the own evaluation function, and (3) imperfect opponent move prediction. We discuss them briefly below in this order.

OM search assumes perfect knowledge of the opponent. It does not take into account any uncertainty about the opponent. (In [5,7] an extension of OM search is described that does include uncertainty: *Probabilistic* OM search.) Since perfect knowledge of the opponent is hardly available in reality, this is too strong an assumption. When the knowledge of the opponent is not perfect, the algorithm can still be used, but this will cause a certain amount of risk, depending on the quality of the knowledge. This first kind of risk has been described extensively in [12].

In [5,8], the second kind of risk in using OM search is investigated. It appears that even when MAX has perfect knowledge of MIN's evaluation function, using OM search may be unwise. For instance, when MAX makes only one large overestimation of the profitability of a certain position while MIN is assessing it correctly, then MAX is possibly attracted to that position. (MAX knows what MIN thinks, but MAX's evaluation of the position is wrong. However, both sides aim for that particular position. This causes an *attractor* in the search tree [8].) A condition that should prevent this from happening is called *admissibility* of the pair of evaluation functions: MAX should not overestimate a position that is overestimated by MIN too.

In this article we introduce a third kind of risk in using OM search: perfect knowledge of the opponent's evaluation function is not equal to perfect prediction of the opponent's moves. This is caused by the difference (normally of one ply) in search depth between a player's prediction of the opponent's move and the actual search that the opponent uses at the next move.

In order for OM search to be successful, the effects of the three types of risks should be alleviated. In a series of experiments with the game of Bao, we investigate what the influence is of a good prediction of the opponent's moves and what the influence is of a better judgement of positions. We assume perfect knowledge of the opponent's evaluation function but admissibility is not guaranteed. Before we describe the experiments, we first refer to the Appendices A and B for an introduction to the domain in which the experiments take place.

6. Generating evaluation functions for Bao

For the OM-search experiments in Bao, we generated five different evaluation functions. It was not our goal to create the best evaluation function possible. Since we presupposed that the quality of evaluation functions influences the performance of OM search, we aimed at creating a set of evaluation functions with different degrees of quality. We used two different approaches of producing them: hand-made and machine-made. The hand-made functions were intended to serve as low-quality functions and as a 'sparring partner' for the machine-made functions. The latter were produced in a number of rounds in which the computer tried to learn an evaluation function that performed best against one particular opponent. By subsequently providing a better opponent, the quality of the obtained evaluation functions was expected to increase. We did not put much effort in optimizing the machine-learning process because the evaluation functions obtained adequately served our purpose. Therefore, the quest for a strong evaluation function for Bao is still open.

The first two evaluation functions were created by hand. The first one, called MATERIAL, simply takes the difference in the number of stones on both sides of the board as the evaluation score.

The second hand-made evaluation function is the one used in the first version of our Bao program and is therefore called DEFAULT. This function incorporates some rudimentary strategic knowledge of Bao. For instance, we incorporated the heuristic rule: it is good to have more stones in your back row since this increases the mobility in the second stage of the game. So, the function awards 3 points to stones in the front row, 5 points to stones in the back row, and 5 additional points to opponent stones that can be captured. If the own house is still active, 200 extra points are given. The total score of the position is the score for MAX minus the score for MIN. There is a small asymmetry in this function: if MAX can capture MIN's house 100 points are rewarded, but if MIN can capture MAX's house, only 50 points are subtracted. This asymmetry is intended to produce a more offensive playing style.

The third evaluation function was created by using a genetic algorithm [11]. The evaluation function was represented by an integer-valued chromosome of 27 genes: one gene for the material balance, one gene per hole for the material in the own back and front row (16 in total), one gene per hole in the front row for capturing (8 in total), one gene for an active house, and another gene for capturing the opponent's house. The total score of a position was defined as the score for the player minus the score for the opponent. The fitness of a chromosome was measured by the number of games (out of 100) that it won against a fixed opponent. In these matches, both players used α–β search with a search depth of 6 ply. The genetic-algorithm parameters were as follows: the population size was 100, only the 10 fittest chromosomes produced offspring (using a single-point crossover), the mutation rate was 5% for large changes in a gene (i.e., generating a new random number for the gene) and 20% for minor changes (i.e., altering the value of a gene slightly). The genetic algorithm was continued until no improvement occurred anymore. We conducted three runs: in the first run, the opponent was DEFAULT. In the second and third run, the opponent was the winner of the previous run. The name of the resulting evaluation function is GA3.

To create the fourth evaluation function, we used TD-Leaf learning [1]. It is a temporal-difference method that is particularly suitable for learning evaluation functions in games. The evaluation function trained was a linear real-valued function with the same parameters as the genes in the chromosomes above, except that there were separate parameters for the two sides of the board. Batch learning was applied with 25 games per batch. The reinforcement signal used to update the parameters was the number of games in the batch that the player won against a fixed opponent, in this case GA3. The search depth used in the games was 10. The λ-factor and the annealing factor both were set to 0.99. The resultant evaluation function was called TDL2B.

The fifth evaluation function was also produced by TD-Leaf learning. This time we used a *normalized Gaussian network* (NGN) as evaluation function, similar to the way in which Yoshoika et al. [20] trained an evaluation function for the Othello game. The NGN had 54 nuclei in a 54-dimensional space. Every dimension correlated with a parameter in the previous evaluation function. The reinforcement signal was the number of games out of 25 won against a fixed opponent, being TDL2B. The search depth used in the games was 6, because the computation of the output for an NGN is relatively slow. No batch learning was applied here. The λ-factor was set to 0.8 and the annealing factor was set to 0.993. This evaluation function is called NGND6A.

The types of the five evaluation functions are as follows: the first two hand-made functions, MATERIAL and DEFAULT, are (possibly rather weak) *profitability* estimators. The other three functions, GA3, TDL2B, and NGND6A, are *win probability* estimators because the fitness functions used involved the number of games won.

7. Experimental setup

To investigate our research questions, we conducted eight different Bao tournaments between five players that each used one of the five evaluation functions. (We denote the players by the name of their evaluation function.) All tournaments followed a double round-robin system: every player was matched against every other player, one time playing South and one time playing North. Furthermore, each player was matched against itself. Each match between two players consisted of 100 games; hence each tournament counted 2500 games. The games began at the 100 start positions given in the appendix and were played to the end. The limit for moves to be denoted as infinite (see rule 1.5a in Appendix B) was set at 100 stones. In the experiments below, moves of 100 or more stones appeared on average in 0.015% of the visited positions of a search tree.

In the first two tournaments, both players used α–β search. These tournaments were intended as a base case. In the other six tournaments, South used OM search with perfect knowledge of the opponent's evaluation function. North always used α–β search. The search depth of South could differ per tournament, but North always used a search depth of 6. No time restrictions were given. Since in Bao draws are not possible (see Appendix B), and since we aimed at comparing the performance of the different search algorithms as used by South, the score of a match was defined as the number of games out of 100 that was won by South.

At every position at which South was to move, we additionally detected the move(s) that α–β search with search depth 6 would select for South. In this way we were able to count the number of times that the moves found by OM search differed from the α–β moves. As a base case, we did the same in the second tournament in which South used α–β search with depth 8.

For completeness we mention a few implementation details of the version of OM search used in the experiments. We opted for an implementation of OM search with α–β probes since this allows for a convenient way to change the level of prediction of the opponent's moves. Furthermore, we used OM search with a restricted speculation of one ply. This means that only in min nodes at depth 1 of the search tree, an α–β probe is performed to predict the opponent's move. This restriction is added because we wanted to remove the risk of inadequate predictions at deeper nodes in the tree and concentrate on the effect of prediction quality in depth-1 nodes.

Some of the evaluation functions used in the experiments are asymmetric. Here, asymmetric means that evaluating a position when South is MAX, is not the same as evaluating the same position when North is MAX and taking the negative of the value. This is especially important during the prediction of the opponent's move by the α–β probes. Therefore, during the probes we temporarily switched the role of the players so that the root of each probe was a max node.

To improve the performance of OM search where possible, we added a provision to deal with multiple equipotent moves for MIN: if MIN has multiple equal choices, MAX will select the move with the lowest value for v_0.

The exact setup of each of the eight tournaments will be explained along with the results in the next section.

8. Results and discussion

First tournament: *α–β plain*. The first tournament was both intended as a base case and as an test of the relative quality of the five evaluation functions. In this tournament, both South and North used α–β search with a search depth of 6 ply. The process that led to the five functions was expected to result in the following operational ordering of evaluation functions: DEFAULT < GA3 < TDL2B < NGND6a. Moreover, since the function MATERIAL is very rudimental in comparison to DEFAULT, we expected that DEFAULT would perform better than MATERIAL. The self-play matches could indicate whether South or North had an overall advantage in the 100 start positions.

Table 1 gives the outcome of the first tournament. The table clearly shows that the evaluation functions differ in quality and that every evaluation function is indeed operationally better in this setting than any of the evaluation functions above them in the table. The difference between MATERIAL and DEFAULT is the smallest but still significant. (Since the size of each match is 100, the 95% confidence intervals are approximately ± 10 per match and ± 20 for the total scores.) The outcome of the self-play matches (the numbers on the diagonal of the table: 54, 43, 53, 45, 43, and 238 on total) provide no statistical evidence of an advantage for either South or North.

Second tournament: *α–β extended*. The second tournament was intended as a checking tournament too. This tournament acted as a reference point to indicate how much could be gained by 2 plies of additional search. Therefore, South was allowed to search two extra plies (8 instead of 6). We expected that all players would gain from the additional search. If not, the evaluation functions would be too weak.

The results are presented in Table 2. The table shows that all players indeed profited from the increased search depth. Only the match of DEFAULT against GA3 (in bold) was less fortunate for DEFAULT. Against the other players, DEFAULT's performance increased. Nevertheless, it illustrates the poor quality of this evaluator.

Third tournament: *OM plain*. This tournament was intended to test the performance of plain OM search in comparison to α–β search. In this tournament, South used OM search with a search depth of 6 ply. The outcome of this tournament would be determined by a balance between the risks of using OM search and by the gain of using perfect knowledge of the opponent. We expected that if the results would be positive, then the better results would be the best for the combinations of high quality against poor quality.

The results in Table 3 show that in summary three players, MATERIAL, GA3, and TDL2B, profited from using OM search, but that the two other players, DEFAULT and NGND6A, did not profit and played worse than in the first tournament. The weakest evaluation function, MATERIAL, improved against all opponents; the strongest evaluation function NGND6A improved hardly against TDL2B and performed worse against all others. The result is inconclusive on the overall performance. Moreover, the tournament refuted our hypothesis that the results should improve with larger difference in quality of the evaluation functions. The auto-play results (diagonal) show that only MATERIAL could substantially profit from using OM search.

The outcome of the third tournament showed that plain OM search with search depth 6 could not outperform α–β. So, in a series of subsequent four tournaments, we tried to find out what additional resources are needed to make OM search play better.

Table 1
Results of the first tournament between 5 evaluation functions. Both players use α–β with a search depth of 6 ply. Each cell shows the number of games won by South (the row) against North (the column). The column on the right shows the number of games won by each evaluation function when playing South

S\N	MATERIAL	DEFAULT	GA3	TDL2B	NGND6A	Score
MATERIAL	54	55	35	19	18	181
DEFAULT	48	43	54	30	28	203
GA3	55	61	53	36	30	235
TDL2B	69	65	57	45	39	275
NGND6A	79	73	75	60	43	330

Table 2
Results of the second tournament between 5 evaluation functions. Both sides use α–β, but South searches 2 ply more deeply (8) than North (6)

S\N	MATERIAL	DEFAULT	GA3	TDL2B	nGND6A	Score
MATERIAL	66	57	62	40	32	257
DEFAULT	71	69	**52**	49	34	275
GA3	80	75	69	62	49	335
TDL2B	86	76	69	60	57	348
NGND6A	88	76	80	70	56	370

Table 3
Results of the third tournament between 5 evaluation functions. South uses OM search with perfect knowledge of the opponent's evaluation function. The search depth is 6 ply for both sides

S\N	MATERIAL	DEFAULT	GA3	TDL2B	nGND6a	Score
MATERIAL	64	57	50	30	24	225
DEFAULT	46	47	46	26	25	190
GA3	59	57	58	40	35	249
TDL2B	78	64	60	52	46	300
NGND6A	71	58	66	61	52	308

Table 4
Results of the fourth tournament between 5 evaluation functions. South uses OM search with perfect knowledge of the opponent's evaluation function. The search depth is 8 ply for South, with α–β probes to depth 6, and the search depth is 6 ply for North

S\N	MATERIAL	DEFAULT	GA3	TDL2B	NGND6A	Score
MATERIAL	68	60	64	49	39	280
DEFAULT	63	58	47	44	41	253
Ga3	70	66	67	57	40	300
TDL2B	80	69	70	55	56	330
NGND6A	84	68	71	59	59	341

Fourth tournament: *OM extended*. In the fourth tournament, we concentrated on the effect of searching more deeply for the max player. A larger search depth for MAX means a better own judgement of the positions and thereby a lower risk of overestimation or underestimation. We expected better results than the outcomes of the second tournament (α–β 8 versus 6).

South was using OM search with 8 ply search for MAX, but α–β probes were still restricted to a depth of 6 ply. This means that South had better knowledge over the game than North, a situation that is comparable to the second tournament. Table 4 shows that South was not able to profit fully from the extra search depth. Although all players performed better than in the third tournament where they were restricted to a search of 6-ply deep, only MATERIAL played better than in the second tournament. The outcome is comparable to the relation between plain OM search and α–β search in tournaments 1 and 3. It indicates that searching more deeply for MAX in OM search is not sufficient for success. The result of the auto-play matches on the diagonal also show results similar to those of tournament 2. The performance of DEFAULT even decreased considerably in comparison.

Table 5
Results of the fifth tournament between 5 evaluation functions. South uses OM search with perfect knowledge of the opponent's evaluation function. The search depth is 6 ply for both sides, but South uses α–β probes to depth 7

S\N	MATERIAL	DEFAULT	GA3	TDL2B	NGND6a	sCORE
MATERIAL	61	59	57	31	27	235
DEFAULT	50	64	48	32	23	217
GA3	53	64	57	36	30	240
TDL2B	69	73	66	58	40	306
NGND6A	77	82	77	63	48	347

Table 6
Results of the sixth tournament between 5 evaluation functions. South uses OM search with perfect knowledge of the opponent's evaluation function. The search depth is 7 ply for South and 6 for North

S\N	MATERIAL	DEFAULT	gA3	tDL2B	NGND6A	Score
MATERIAL	75	69	60	46	33	283
DEFAULT	64	61	68	48	34	275
GA3	75	71	69	46	38	299
TDL2B	82	81	81	62	49	355
NGND6A	88	80	82	66	**51**	367

Fifth tournament: *OM with perfect opponent prediction*. Part of the poor performance of OM search could be caused by the quality of the prediction of the opponent's move. OM search with depth 6 uses α–β probes that are effectively 5 ply deep. The true opponent uses a 6-ply search, so there is a good chance that the probes will not predict the opponent's move correctly, even given that South uses the true evaluation function of the opponent. Therefore, in the fifth tournament South was allowed to extend the α–β probes to depth 7—and thus searched to the same depth as the opponent. In the case of equal evaluated (equipotent) moves, South selected the move with the lowest own evaluation. This was not necessarily the move that North would play. The search depth for MAX was 6 ply like in tournament 3.

We clearly expected the results to be better than those of tournament 3 since part of OM search's risk was eliminated. Table 5 gives the results of this tournament. All players, except Default profited from this advantage, and played better than in tournament 1, albeit less good than in the second tournament in which they just searched more deeply. The advantage also gave less good results than the advantage in tournament 4, except for player NGND6A. fROM THESE RESULTS we may infer that knowing exactly the moves of the opponent does not help if the own judgement is too weak.

Sixth tournament: *OM plain, depth* 7. Since neither the extended search for MAX nor the enlarged α–β probe could improve the performance of OM search sufficiently, the next step was to combine both. Of course, it is desirable to reach a good result with as few additional resources as possible, therefore we started with the combination of search depth 7 for MAX and α–β probes to depth 7. Effectively, this is equal to applying plain OM search with search depth 7.

The results in Table 6 show that in all matches but one (in bold) South performed better than in tournament 3 where OM search with depth 6 was used. Moreover, the performance was in all matches better than in tournament 1, where α–β search with depth 6 was used. However, the results of tournament 6 were not as good as the results of tournament 2 (α–β search with depth 8). It means that OM search with one extra ply search could outperform α–β, but could not beat α–β search with an additional two-ply search.

Seventh tournament: *OM perfect*. In the seventh tournament we took OM search one step further and gave MAX one additional ply to search. The tournament combined the advantages of the fourth and fifth tournament for South: the search depth for the own evaluation was 8 ply for South and the α–β probes for the opponent extended to depth 7. The results in Table 7 show that the power of South had significantly increased. Just as in the sixth tournament, all players performed better than in tournament 1, and all players, except GA3 also played better than in tournament 2. The results of Ga3 were only slightly inferior to those in tournament 2.

When we observe the improvement of the scores in Table 7 in relation to those in Table 1, it appears that the strongest improvements occur with the weakest player against the stronger players: there is even a difference of 40 between the two matches MATERIAL versus NGND6A. The next best performance improvement is measured with the player TDL2B. The lowest improvement (only 1 point) is between the two matches NGND6a Versus MATERIAL. This means a rejection

Table 7
Results of the seventh tournament between 5 evaluation functions. South uses OM search with perfect knowledge of the opponent's evaluation function. The search depth is 8 ply for South, with α–β probes to depth 7, and the search depth is 6 ply for North

S\N	MATERIAL	DEFAULT	GA3	TDL2B	NGND6A	Score
MATERIAL	73	76	69	54	58	330
DEFAULT	59	76	66	48	46	295
GA3	75	77	79	56	55	342
TDL2B	79	88	83	70	57	377
NGND6A	80	88	85	68	63	384

Table 8
Results of the eighth tournament between 5 evaluation functions. South uses OM search with perfect knowledge of the opponent's evaluation function and strict risk avoidance. The search depth is 6 ply for both sides

S\N	MATERIAL	DEFAULT	GA3	TDL2B	NGND6a	Score
MATERIAL	67	66	49	32	33	247
DEFAULT	49	48	45	37	31	210
GA3	63	62	54	35	33	247
TDL2B	72	66	64	58	46	306
NGND6A	75	71	76	69	47	338

of the hypothesis that a large difference between the evaluation functions contributes to the performance of OM search. On the contrary, it appears that OM search was the most profitable for the weakest player. Of course, MATERIAL was still beaten by the three best players in tournament 7, but by a smaller margin than in tournament 1. Moreover, the improvements seem to increase with the strength of the opponent's evaluation function, but there is no strong statistical evidence for this.

Eighth tournament: *OM with strict risk avoidance*. The results of the previous four tournaments indicated that it was necessary to increase both the search depth for MAX with 2 ply and to increase the depth of the probes with one ply before OM search could outperform α–β search. In the tournament, we wanted to test another strategy, namely to avoid risk as much as possible. In this tournament, South applied OM search but only deviated from the strategy that α–β search imposed in case the move that OM search advised had the same Minimax value. It means that an additional full-depth α–β search was needed. The search depth was equal to that of the third tournament.

Since it occurs relatively often in Bao that multiple moves at the same position have the same Minimax value, we expected that South did have some room to speculate and expected that this approach could outperform α–β search and also plain OM search. We did not expect this method to outperform α–β search with a search of depth 8.

The results in Table 8 show that the approach was indeed successful. All players had a better overall score than in the first tournament. They all performed better (or equal in case of TDL2B) than in the third tournament, too. As expected, the results were not as good as in the second tournament.

A summary. Table 9 summarizes the results of the eight tournaments. On all rows the scores are increasing from left to right (except for the first two columns). This means that the order of quality for the evaluation functions, as observed in the experiments, is as follows: (MATERIAL, DEFAULT) < GA3 < TDL2B < NGND6a. The ordering of MATERIAL and DEFAULT is unclear: this is not a surprise, since both evaluation functions are poor. The table shows that if only the search depth is increased (4: OM extended) or only the prediction of the opponent is improved (5: OM perfect knowledge of opponent), the results are not as good as just using α–β with extra search depth. When both methods were combined, (6: OM plain depth 7; and 7: OM perfect), the results are better. Furthermore, the table shows that using OM search with strict risk avoidance (8: OM no risk) leads to better results than using plain OM search or plain α–β.

Deviations. The overview of Table 10 provides insight into the number of times that OM search deviated from the α–β search strategy. As a reference we also measured the number of times that α–β search with search depth 8 deviated from α–β search with search depth 6. The table shows that searching more deeply for the own evaluation had a larger effect than searching more deeply for the opponent prediction: the results for the tournaments 2, 4, and 7 are comparable. Moreover, the table shows that the number of deviations was larger in tournament 4 than in tournament 7. Since the results of tournament 4 were inferior to those of tournament 7, it seems that an incorrect prediction of the

Table 9
Overview of the eight Bao tournaments

Tournament	d	p	MATERIAL	DEFAULT	GA3	TDL2B	NGND6a
1: α–β plain	6	–	181	203	235	275	330
2: α–β extended	8	–	257	275	335	348	370
3: OM plain	6	6	225	190	249	300	308
4: OM extended	8	6	280	253	300	330	341
5: OM perf. opp.	6	7	235	217	240	306	347
6: OM plain	7	7	283	275	299	355	367
7: OM perfect	8	7	330	295	342	377	384
8: OM no risk	6	6	247	210	247	306	338

Table 10
Overview of the average number of times per game in which the move that OM search selects differs from the move that α–β search (with a search depth of 6 ply) suggests. The standard deviation ranges between 0.5 and 2.5. The average number of moves per game for South is 19.9 over all games in the tournaments

Tournament	MATERIAL	DEFAULT	GA3	TDL2B	NGND6A
2: α–β extended	4.58 ± 2.4	5.19 ± 2.3	4.23 ± 2.2	3.73 ± 1.9	3.86 ± 2.0
3: OM plain d6	3.05 ± 2.1	2.90 ± 1.9	2.13 ± 1.7	2.38 ± 1.6	2.13 ± 1.5
4: OM extended	4.79 ± 2.5	5.55 ± 2.4	4.76 ± 2.4	4.47 ± 2.2	4.55 ± 2.3
5: OM perf. opp.	3.30 ± 2.0	3.22 ± 1.9	2.50 ± 1.6	2.32 ± 1.6	2.24 ± 1.5
6: OM plain d7	2.29 ± 1.8	2.30 ± 1.8	1.84 ± 1.6	1.92 ± 1.5	1.82 ± 1.5
7: OM perfect	4.38 ± 2.2	5.15 ± 2.4	4.49 ± 2.2	3.93 ± 2.0	3.97 ± 1.9
8: OM no risk	1.91 ± 1.6	0.86 ± 1.1	0.31 ± 0.5	0.76 ± 0.9	0.49 ± 0.7

opponent leads to extra deviations that do not contribute to a positive outcome. The lower values for tournament 8 can be explained by the risk avoidance.

9. Conclusions and future research

Below we answer the five research questions (Section 9.1), then we provide an overall conclusion (Section 9.2). Finally, we give some ideas for future research (Section 9.3).

9.1. Answers to the five research questions

The first research question was: how can evaluation functions be ordered? This is an important question in the context of OM search. We showed that evaluation functions can be arranged according to three types: win predictors, probability estimators, and profitability estimators. Moreover, we introduced eight possible orderings of evaluation functions. Only the operational ordering of evaluation functions is applicable to all types of functions. This ordering is mostly used in practice. So, the answer is that the evaluation functions can be ordered practically according the operational ordering.

To answer our further research questions, we provided some theoretical exercises. Moreover, we performed a large experiment consisting of eight Bao tournaments. The second research question stated: to what extent does the quality of the prediction influence the performance of OM search? The outcome of the tournaments provided clear evidence that enhancing the prediction by one additional ply of search in the predictive probes improves the performance of OM search. However, one additional ply of search turned out not to be sufficient to outperform α–β search.

Our third research question was: does the relative quality of two evaluation functions influence the performance of OM search? We hypothesized that it was best to have a high-quality function versus a weak-quality functions. The experimental results refuted this hypothesis. Yet, the experiments indicated that some evaluation functions might profit more than others from using OM search.

The fourth research question was: does the quality of the own judgements influence the performance of OM search? The results of the tournaments gave rise to a positive answer. However, also the additional two plies of search for MAX were not sufficient to outperform α–β search.

A combination of both deeper probes and deeper search was able to outperform α–β search. So our answer to the third and fourth research question is: a definite yes, but two factors needed to be combined before the total improvement was significant.

Then, the additional resources granted to OM search should be compared to the case in which α–β search was granted additional search depth, which was our fifth research question: can OM search with the additional search depths outperform α–β with the same additional search depths? The experiments provided evidence for a positive answer to this question.

9.2. Overall conclusion

The experiments were not exhaustive and therefore did not produce firm qualifications although many effects may be considered statistically significant. The tournaments gave a good insight into the working of OM search and the role that evaluation functions play in it.

The experiments were performed in Bao. Yet, the results can be generalized to other games. The consequences of the results for OM search in general are that the search method only can be applied successfully when additional resources (e.g., search time) are available. Either additional search time (in comparison with the opponent) must be spent for the prediction of the opponent's move and the extended search; or additional search time must be spent to perform risk management. If these additional resources are not available, OM search cannot be applied successfully.

To conclude, we provided a theoretical discussion on interpreting and ordering evaluation functions that can be of use for selecting evaluation functions in general. The experimental results produced unexpected results concerning the selection of evaluation functions in OM search. More detailed research is needed to identify the causes of these results.

9.3. Future research

In order to measure the effects of opponent prediction and extended search more precisely, the sample size should be increased and more game details should be analyzed, such as the number of times that the predicted move differs from the move played by the opponent. Furthermore, a deeper study of the properties of the trained evaluation functions and some matches between players might provide more background information. A final suggestion for future research is to investigate the possibilities for risk management more extensively since this seems necessary as well as promising.

Acknowledgements

We would like to thank the anonymous referees who provided us with substantial comments and constructive ideas. They allowed us to improve the paper considerably.

Appendix A. Mancala games

In large parts of the world, board games of the mancala group are played in completely different versions (cf. [13,15]). Whatever the case, most mancala games share the following five properties:
(1) the board has of a number of *holes*, usually ordered in rows;
(2) the game is played with indistinguishable *counters* (also called pebbles, seeds, shells);
(3) players own a fixed set of holes on the board;
(4) a move consists of taking counters from one hole and putting them one-by-one in subsequent holes (*sowing*), possibly followed by some form of a capture;
(5) the goal is to capture the most counters (for Bao it is to immobilize the opponent).

Mancala games differ in the number of players (1, 2, or more), the size and form of the board, the starting configuration of the counters, the rules for sowing and capturing, and in the way the game ends. The games of the mancala group are known by many names (for instance Kalah, Awari, Wari, Awele, Bao, Dakon, and Pallankuli). For an overview of different versions and the rules of many mancala games, we refer to [15].

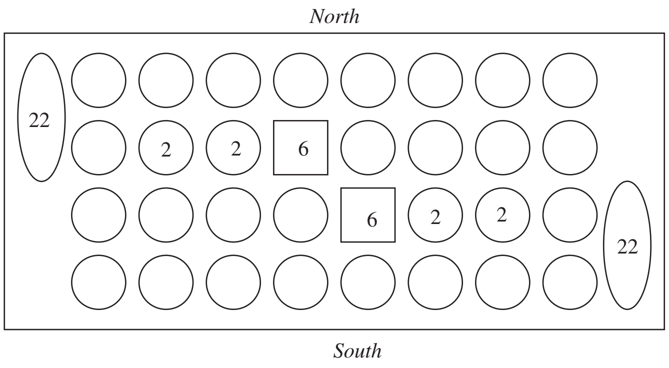

Fig. 3. Opening position of Bao.

Among the mancala games, (Zanzibar) Bao is regarded as the most complex one [19]. This is mainly due to the amount of rules and to the complexity of the rules. Bao is played in Tanzania and on Zanzibar in an organized way. There exist Bao clubs that own boards and that organize official tournaments.

The exact rules of the game are given in Appendix B. Below, we summarize the properties that discriminate the game from the more widely known games Kalah and Awari.

Bao is played on a board with 4 rows of 8 holes by two players, called South and North, see Fig. 3. Two square holes are called *houses* and play a special part in the game. There are 64 stones involved. At the start of the game each player has 10 stones on the board and 22 stones in store. Sowing only takes place in the own two rows of holes. The direction of sowing is not fixed. At the start of a move, a player can select a direction for the sowing (clockwise or anti-clockwise). During sowing or at a capture, the direction can turn at some point. This is regulated by deterministic rules.

In Bao, one is obliged to capture, if possible. It means that a position is either a capture position or a non-capture position. Captured counters do not leave the board but re-enter the game. Counters are captured from the opponent's front row. The captured counters are immediately sown in the own front row. It implies that the game does not converge like Kalah and Awari.

Moves are composite. For instance, at the end of sowing stones after a capture, a new capture might be possible. The second capture is executed and the captured counters are again sown immediately at the own side of the board. In its turn, the second sowing can lead to a third capture, followed by a third sowing, etc. If a capture is not possible, and the hole reached was non-empty, all counters are taken out of that hole and sowing continues. This procedure stops when an empty hole is reached, which ends the move.

Moves can be endless because in a non-capture move, sowing may continue forever. The existence of endless moves can be proven theoretically [9]. [In real games, moves that take more than an hour of sowing also occasionally occur, but players usually make small mistakes during sowing or simply quit the game. So, real endless moves never lead to endless sowing, in practice (see rule 1.5a in Appendix B).]

Bao games consist of two stages. In the first stage, stones are entered one by one on the board at the start of every move. In that stage, a game ends if the player to move has no counters left in the front row. As soon as all stones are entered, the second stage begins and a new set of rules applies. In this stage, a game ends if the player has no more than one counter in any hole of both rows. A draw is not defined in Bao. Note that the goal of Bao is not to capture the most stones, but to immobilize the opponent.

We selected this game because it has a fairly narrow game tree: the maximum number of moves possible at any position is 32, but the average number of possible moves varies between 3 and 5. The narrow game tree allows for a reasonably deep search (6 to 8 ply) and a large number of games (22,500 matches of 20 moves each on average).

In [6], a more detailed analysis of the game properties of Bao is provided. The state-space complexity of Bao is approximated to be 1.0×10^{25}, which is much higher than that of Awari (2.8×10^{11}) and Kalah (1.3×10^{13}). The shortest game possible takes 5 ply, but most games take between 50 and 60 ply. Forced moves occur quite often. The average game length (d) and branching factor (w) are normally used to estimate the size of a game tree (w^d).

For Bao the estimate is roughly 10^{34}. This number together with the state-space complexity (10^{25}) places Bao in the overview of game complexities above Checkers and in the neighbourhood of Qubic [10].

Appendix B. Zanzibar Bao rules for the computer

Below we provide a version of the rules for Zanzibar Bao especially adapted for computer use. These rules are based on De Voogt's [19] thesis.

The board

Zanzibar Bao (or Bao for short) is a game from the mancala family (see [19]). It is played by two persons on a board with four rows of eight pits or holes, see Figs. 3 and 4. The two lower rows are owned by the player called South, the two upper rows are owned by the player called North. The two middle rows are called front rows and the two outer rows are called back rows. The fifth pit from the left (in perspective of the player) on each front row is shaped differently and is called the *nyumba* or house. The two outer pits of the front row are called *kitchwa*, and the two pits on the front row next to the kitchwa are called the *kimbi*. Bao is played with 64 equal counters (seeds, stones, or *kete*).

The front row of South is indicated by the character ‘A’, the back row is indicated by ‘B’. For North, ‘a’ and ‘b’ are used for the front row and back row, respectively. The holes on each rows are numbered from 1 to 8, starting at the left hand of the player owning the rows, see Fig. 4. South’s nyumba is indicated by ‘A5’ and North’s by ‘a5’. The kitchwas are: A1, A8, a1, and a8, the kimbis are: A2, A7, a2, and a7.

A configuration of stones on the board is indicated by four rows of numbers, or two rows if the back rows are not of interest, in the same pit order as in Fig. 4. The rows of North are always on top. A wildcard ‘*x*’ can be used to indicate that the content of a non-empty pit is not of interest or not known.

General rules

Rule 1.1: *Goal of the game*. The goal of Bao is to empty the front row of the opponent or to make it impossible for the opponent to move.

Rule 1.2: *End of the game*. The game ends if (1) the front row of a player is empty (even during a move) or (2) if a player cannot move. In both cases the other player wins.

Rule 1.3: *Sowing*. The basic move of Bao is sowing (spreading) of stones. Sowing is the process of putting a determined number of stones one by one in consecutive holes in the own two rows of the board in clockwise or anti-clockwise direction. During sowing, the direction of the sowing cannot change. Every sowing (spread) has a starting pit, a number of stones to sow, a sowing direction, and an ending pit.

Rule 1.4: *Single capture*. Capturing in Bao is allowed only if a sowing ends in a non-empty pit at the front row that has an opposing non-empty pit at the front row of the opponent (this is called *mtaji*). The player’s pit is called the *capturing pit* and the opponent’s pit is called the *captured pit*. The capture in Bao consists of taking all stones out of the captured pit in the opponent’s front row and sowing them immediately at the own side of the board. The sowing of captured stones must start at one of the kitchwas.

1.4a: If the sowing starts at the left kitchwa, the sowing direction is clockwise, if the sowing starts at the right kitchwa, the sowing direction is anti-clockwise.

1.4b: If the capturing pit is the left kitchwa or kimbi, the sowing must start at the left kitchwa. If the capturing pit is the right kitchwa or kimbi, the sowing must start at the right kitchwa.

1.4c: If the capturing pit is not a kitchwa or kimbi, then the direction can be chosen by the player if the capture is the start of a move in namua stage (see below). Otherwise the existing direction of the move must be sustained.

Rule 1.5: *Move*. A move in Bao is a sequence of sowings and captures by one player. A move stops if a sowing ends in an empty pit, but may also stop at the house (see rule 2.5b) or at a takasia-ed hole (see rule 4.1b). After a move the opponent is to move.

Rule 1.5a: *Infinite moves* (*special computer rule*). A move can take a long time and sometimes last forever. However, these infinite moves are illegal. If no finite move is available, the game is lost for the player to move. Because infinity of a move can be very tedious to prove, a move is regarded infinite if more than a previously designated number of stones is sown.

Rule 1.6: *Endelea*. If a sowing ends in a non-empty pit (e.g., after sowing there are more than one stone in the ending pit) and a capture is not allowed, then the move continues in the same direction by taking all the stones from

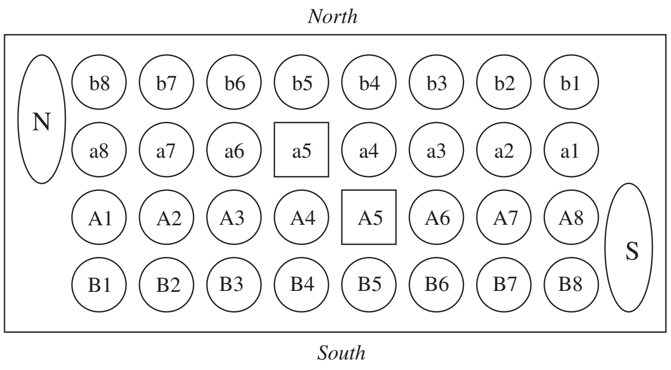

Fig. 4. Numbering of the holes in Bao.

that pit and sowing the stones starting at the next pit in the same direction. This continuation of sowing is called *endelea*.

1.6a: Endelea stops if the sowing ends at the owned house that contains six or more stones if the player does not (or cannot) decide to play the house (see rule 2.5b).

1.6b: Endelea stops if the sowing ends at a takasia-ed hole.

1.6c: Endelea stops if a capture is possible. The move continues with the capture. The direction of the sowing of captured stones is the same as the direction of endelea, unless capture occurs at the kichwa or kimbi.

Rule 1.7: *No-capture moves* (*takasa*). If a move does not start with a capture, then capturing is not allowed at all during that move. The player is then called to *takasa*. During takasa, the player keeps performing endelea until it ends (rule 1.6a/b). During takasa, the direction of the move cannot change.

Rule 1.8: *Capture moves*. If a move starts with a capture, then more captures can occur during endelea later on. If captures take place at the kitchwa or kimbi, the direction of the moves changes.

1.8a: It is obligatory to capture, if possible.

Rule 1.9: *Stages*. There are two stages in Bao: *namua* and *mtaji*.

Namua stage

Rule 2.1: *Start of namua stage*. The game starts in namua stage with the following board configuration (see Fig. 3): there are six stones in South's nyumba and two stones in the hole to the right of the nyumba and again two stones in the next hole to the right. The same number of stones are in North's nyumba and in the consecutive holes to the right (of North). Each player has 22 stones in store. The first player is South.

Rule 2.2: *Move*. During namua stage, the player starts each move with *sowing one stone from the store* on the board. When all stones are on the board (after 22 moves or 44 plies) the game enters the mtaji stage.

Rule 2.3: *Capturing in namua stage*. A player is allowed to capture a non-empty opponent's hole from the front row if there is a non-empty hole at the own front row (the capturing hole), directly opposite to the captured hole. The player puts ('sows') one stone from the store in the capturing hole, takes all stones from the opponent's captured hole and start sowing them at one of the own kichwas as described in rule 1.4.

2.3a: If the capturing hole is not a kichwa or kimbi, the player may choose which kichwa to sow from. The move continues as described above.

2.3b: If a player can capture, he must do so.

Rule 2.4: *Takasa in namua stage*. If a player cannot capture, he must takasa. In namua, takasa starts with sowing one stone from the store in a non-empty hole in the front row. Takasa cannot start in the back row.

2.4a: If the only filled hole on the front row is one of the kichwas, then takasa cannot be done in the direction of the back row (because the front row will be empty and the game is a loss).

2.4b: Takasa cannot start from the owned house with six or more seeds unless it is the only filled hole in the front row. If it is the only hole filled at the front row, then one stone from the store must be put in it, *two* stones have to be taken out and spread to the left or to the right.

2.4c: Takasa cannot start from a hole with only one stone, unless there are no holes with more than one stone on the front row or unless the house is still owned (even with fewer than 6 stones).

```
South: Ramadhan
North: Kijumbe
place: Zanzibar
date: 17-10-94
winner: North
time: 30 minutes 10 seconds
takasia: yes
1: 7L* 5R;
2: 6R* 6R*;
3: 7R* 8L*;
4: 8R* 6R*;
5: 5R* 8L*;
6: 7R* 6R*;
7: 6R* 8L*;
8: 7R* 6R*;
9: 8R* 7;
10: 5L* 5L;
11: 3R 1;
12: 5R 7R*; end in nyumba
13: 8 2;
14: 8 2;
15: 3R* 5R>;
16: 5R 3L;
17: 8 6R;
18: 4R 7;
19: 5R 8;
20: 5R 7;
21: 7 6L;
22: 2 8;
23: B2L b7R;
24: A7L b8R;
25: A3R a6L;
26: A7L a6L; South resigns
```

Fig. 5. A Bao game transcript.

Rule 2.5: *The house* (*nyumba*). At the start of the game, both players own their house.

2.5a: Players lose their house if it is emptied or when the first capture is made in mtaji stage.

2.5b: If the player still owns the nyumba and a sowing ends in the nyumba then (1) the move ends during takasa if the house contains six or more stones, or (2) the move ends during a capture move if no capture is possible at the nyumba and *if the player wishes to stop*. If a player does not wish to stop then the player is said to *play the house*, which means that the house is emptied and its stones are spread.

Mtaji stage

Rule 3.1: *Start of mtaji stage*. The mtaji stage starts as soon as all stones are on the board.

Rule 3.2: *The house*. If the house happens to be still owned by one of the players, it stays owned until the first capture occurs. The namua rules for the house (2.4b, 2.4c, and 2.5b) do not apply anymore. Takasia rules do apply (4.1c) on the house.

Rule 3.3: *Moves*. In mtaji stage, only holes can be played that contain more than one stone. If both rows of a player only contain holes with zero or one stones, this player loses the game. Every move in mtaji stage starts with selecting a hole and sowing the contents in a chosen directory.

Rule 3.4: *Capture move in mtaji stage*. A capture move in mtaji stage must start from a hole on the front row or back row that contains more than one but fewer than 16 stones. After spreading in a chosen direction, the last stone must allow capturing. If a capture is possible, it is obligatory. The direction of the sowing of the captured stones is the same as the selected move direction, unless capture occurs on the kichwa or kimbi.

Rule 3.5: *Takasa in mtaji stage*. If no capture move is possible, the player must takasa.

3.5a: If possible, the player must takasa from the front row.

3.5b: If the only filled hole on the front row is one of the kichwas, takasa cannot go in the direction of the back row (because the front row will be empty and the game is a loss).

Special rule

Rule 4.1: *Takasia*. If a player must takasa, but can play such that (1) the opponent also must takasa next move and (2) exactly one of the opponent's hole can be captured after that, then the opponent is not allowed to empty this *takasia-ed* hole. (This can only happen in mtaji stage.)

4.1a: However, a hole cannot be *takasia-ed* (e.g., the opponent is allowed to empty it) if it is the house, or if it is the only occupied hole in the front row, or if it is the only hole containing more than one stone in the front row.

4.1b: If endelea ends in a takasia-ed hole, the move ends.

4.1c: If a house is still owned during mtaji stage, it cannot be takasia-ed.

Notational system

Moves are indicated by the row (‘A’,‘B’,‘a’,‘b’) and number of the hole from which the move starts (‘1’–‘8’). The direction of the move is indicated by ‘L’ or ‘R’. When a player decides to play the house, a ‘>’ is added to the move. A takasa move is indicated by an asterisk ‘$*$’, a takasia is indicated by two asterisks ‘$**$’.

In namua stage, the row indication can be omitted. If the capturing hole is a kichwa or kimbi, the direction indication can be omitted.

The direction indicator is relative to the player at move. It indicates the direction in which the hand moves after putting the first stone in namua stage or after picking up the stones of a pit in mtaji stage. So, in a capture move during namua stage, the direction indicates whether the left (L) or right (R) kichwa is chosen to be started from.

A game transcript consists of a head containing the game information and one line for every two plies (one move). A move line starts with the move number, then a colon, a space, the move for South, a space, the move for North and a semicolon follow. After the semicolon, comments may be added. An example game transcript is given in Fig. 5.

Appendix C. Bao start positions

The following table gives the 100 start positions that are used in the Bao experiments. The positions are generated by playing 10 random legal moves for every player from the official Bao opening position. Each row gives the contents of the holes of one position. The numbering of the holes is according to Fig. 4. The last two columns indicate whether South and North have an active house.

Row b								Row a								Row A								Row B								House	
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	S	N
1	1	0	0	1	1	1	1	0	0	0	0	1	2	10	0	0	3	9	1	2	4	0	0	0	0	0	0	0	0	1	1	F	F
1	1	1	1	1	1	1	1	1	3	7	0	4	0	6	2	0	1	0	1	0	5	0	0	0	0	0	0	0	0	1	1	F	F
3	2	1	0	0	0	0	0	0	1	1	11	1	0	1	0	0	2	2	0	0	2	4	0	1	1	1	1	1	1	2	1	F	T
0	0	0	0	0	0	0	0	1	5	6	0	1	3	0	1	0	0	1	7	1	1	5	0	1	1	1	1	1	1	0	2	F	F
1	1	0	0	0	1	1	1	0	0	0	1	1	0	0	0	1	0	0	3	15	1	1	5	1	1	0	0	0	1	2	2	T	F
1	0	3	3	1	0	4	1	6	0	1	1	10	5	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	F	F
0	1	1	0	2	1	3	3	0	3	6	6	0	0	0	1	1	1	0	1	0	4	3	0	1	0	0	0	0	0	1	1	F	F
2	2	0	1	1	0	1	1	0	1	0	1	1	1	1	0	1	0	0	8	12	1	0	0	2	1	1	1	0	0	0	0	T	F
1	1	1	1	1	1	1	3	3	1	1	1	0	4	3	1	0	4	1	2	0	6	2	1	0	0	0	0	0	0	0	0	F	F
2	1	0	0	0	1	2	0	0	0	1	3	4	1	0	4	1	1	5	0	2	5	0	0	1	0	0	0	0	1	2	3	F	F
3	0	1	1	0	0	0	0	0	2	0	9	0	0	3	1	1	0	2	0	8	0	0	0	1	1	1	1	0	2	0	3	T	T
1	1	0	0	0	0	1	1	0	1	1	1	6	1	3	2	1	1	5	3	1	2	0	1	2	2	1	0	0	0	1	1	F	F
2	1	1	1	1	1	1	1	1	1	0	1	2	0	5	1	2	1	1	0	0	4	0	3	0	2	0	2	0	2	2	1	F	F
2	2	1	0	0	0	0	0	0	3	2	11	1	0	1	0	0	0	0	1	13	0	0	0	0	0	0	0	0	1	2	0	T	T
0	0	0	1	1	1	1	1	0	0	7	0	1	1	1	0	2	6	0	0	1	0	6	0	0	2	0	2	0	2	1	3	F	F
1	1	0	0	0	0	0	0	1	2	4	14	4	4	1	1	0	0	0	0	1	0	1	0	0	2	1	0	0	0	1	1	F	T
0	2	2	0	1	1	0	0	1	4	2	11	0	0	0	0	0	0	0	0	9	1	1	0	0	0	0	0	1	1	1	2	T	T
0	0	0	0	0	0	1	1	2	2	1	1	2	2	1	1	1	7	0	0	13	0	0	1	1	1	1	0	0	0	0	1	T	F
1	1	0	1	1	1	1	1	1	4	1	1	1	5	1	0	0	0	0	0	13	0	5	0	0	0	0	0	0	0	0	1	T	F
3	1	1	1	0	0	1	1	0	1	0	11	0	0	0	0	0	0	0	0	10	1	1	0	2	0	1	1	1	1	0	2	T	T
1	1	0	0	1	1	1	1	2	4	1	6	0	4	3	3	1	3	6	1	0	0	0	0	0	0	0	0	0	0	0	0	F	F
1	0	2	1	0	2	0	2	0	4	0	0	7	5	0	1	0	0	6	2	0	1	1	1	0	0	0	0	1	1	0	2	F	F
2	2	1	1	1	1	1	1	1	2	2	0	1	2	0	2	0	0	0	0	12	1	0	1	1	1	0	0	0	1	1	2	T	F
0	1	3	0	1	4	4	0	1	5	1	4	2	4	5	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	F	F
1	1	0	0	0	1	1	2	0	3	0	9	0	0	2	1	1	0	0	0	11	0	0	0	0	0	1	1	0	2	0	3	T	T
0	2	2	1	3	0	1	1	6	1	0	2	1	0	1	2	2	1	6	0	1	0	3	3	0	0	0	0	0	0	0	1	F	F
0	0	0	0	0	0	0	0	1	0	3	10	0	5	0	1	1	1	0	0	15	0	2	1	0	0	0	0	0	0	0	0	T	T
1	1	0	2	0	2	2	2	0	1	4	5	7	4	0	1	0	0	0	0	0	1	1	3	0	0	0	0	0	1	2	0	F	F
2	2	2	1	1	1	1	1	2	3	1	3	6	2	1	0	0	0	0	0	0	1	3	0	1	1	1	1	1	0	1	1	F	F
0	0	0	0	0	0	0	1	1	0	8	0	0	0	3	0	0	8	0	4	4	0	2	2	1	1	1	1	0	2	1	0	F	F

2	1	2	2	1	1	1	1	0	1	4	0	1	0	6	1	1	0	0	0	2	9	0	0	0	0	0	0	1	1	0	2	F	F
0	0	0	1	1	1	1	1	0	0	0	6	0	1	1	0	0	9	1	0	2	1	5	0	2	2	0	2	0	2	1	0	F	F
0	0	0	0	0	0	0	0	1	2	1	3	1	3	1	1	1	2	0	0	6	1	0	2	2	0	2	2	2	1	4	2	F	F
1	1	0	1	1	1	1	1	0	5	4	0	5	1	7	0	0	1	0	1	0	6	0	0	0	0	0	0	0	1	2	0	F	F
0	1	1	0	1	1	1	1	3	1	0	0	0	0	0	0	0	4	3	4	5	5	3	2	1	0	0	0	0	1	1	1	F	F
0	2	1	2	0	2	0	2	0	5	1	1	5	0	6	0	0	0	0	3	0	1	1	3	0	2	1	0	0	0	1	1	F	F
0	0	0	0	0	0	0	0	1	3	1	3	1	1	1	1	2	1	4	0	1	3	2	5	2	1	2	0	4	0	1	0	F	F
1	1	1	1	1	0	1	1	0	1	4	0	0	5	0	1	4	2	0	3	1	1	0	6	2	1	0	0	0	0	1	1	F	F
3	0	2	0	2	0	2	2	1	3	1	0	0	0	0	2	1	0	0	0	11	2	0	0	0	0	0	1	1	2	0	4	T	F
1	1	2	0	4	2	4	0	0	6	0	1	9	4	1	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	1	1	F	F
0	0	0	0	0	0	0	0	4	1	1	3	4	0	0	5	1	2	0	0	0	3	3	3	1	1	1	1	1	2	2	1	F	F
1	1	1	0	0	0	0	0	1	0	0	14	0	0	1	0	1	6	0	0	0	1	3	0	2	2	2	2	0	1	1	0	F	T
1	1	1	0	0	0	0	1	0	0	0	1	1	3	0	5	2	1	2	3	2	0	0	0	0	2	1	3	1	3	3	3	F	F
1	1	0	0	0	0	0	0	1	0	2	12	5	4	0	1	0	0	0	0	0	0	3	1	0	1	2	0	2	0	2	2	F	T
0	1	2	0	1	1	0	1	0	1	2	11	0	4	2	0	0	1	1	2	0	0	1	0	2	0	2	0	2	0	1	2	F	T
2	2	2	2	1	0	0	0	0	2	1	3	1	1	3	1	0	1	3	0	10	0	0	1	1	1	0	0	0	0	1	1	T	F
1	1	0	0	0	0	0	0	0	0	0	0	2	6	1	1	1	6	9	4	2	0	1	0	1	1	1	1	1	0	0	0	F	F
0	0	0	0	0	0	0	1	0	0	6	1	2	2	1	1	4	1	3	4	0	0	0	4	3	1	0	1	3	0	1	1	F	F
0	0	0	0	0	0	0	0	0	6	1	1	0	4	3	1	1	0	1	3	15	0	1	0	0	0	0	0	0	1	1	1	T	F
1	0	0	0	1	1	1	1	0	4	2	2	2	0	1	1	0	0	0	3	3	0	8	0	0	2	0	2	0	4	0	1	F	F
1	1	0	0	0	0	1	1	2	1	0	1	2	5	2	0	2	3	3	6	2	0	1	0	1	1	0	0	1	1	0	2	F	F
0	0	0	0	0	0	0	0	0	0	1	1	1	3	0	1	1	5	1	9	4	2	0	0	3	1	2	0	2	2	0	1	F	F
0	0	0	0	0	0	0	0	5	0	3	5	4	0	0	1	0	0	0	0	0	9	3	0	1	3	1	0	1	3	1	0	F	F
1	0	2	0	2	0	2	0	0	3	1	1	5	0	1	1	0	0	3	1	2	1	12	0	0	0	0	0	0	1	1	0	F	F
2	1	1	2	2	0	2	0	0	3	1	1	0	7	0	0	0	0	0	0	15	0	1	0	0	0	0	0	0	0	1	1	T	F
2	1	1	1	0	0	0	0	0	5	0	13	0	0	0	1	0	1	4	0	0	2	2	0	1	1	1	1	1	0	1	1	F	T
2	0	1	1	0	0	0	0	0	4	1	8	0	5	1	0	0	0	0	0	9	0	0	0	0	0	1	1	1	3	0	2	T	T
2	1	2	1	0	1	3	0	0	6	1	5	5	1	3	1	0	0	0	0	0	0	0	3	0	0	1	1	0	2	1	0	F	F
0	1	2	0	1	1	0	0	0	3	1	13	1	0	1	0	0	0	1	0	9	0	0	0	0	0	0	0	1	1	1	3	T	T
2	0	1	0	2	0	2	2	0	3	5	1	3	2	1	1	0	3	0	0	0	0	5	1	0	0	0	1	1	1	1	2	F	F
1	0	3	2	1	0	0	0	0	1	1	10	0	0	0	0	0	0	0	0	12	1	2	1	0	0	0	0	0	1	2	2	T	T
0	2	1	0	0	0	0	0	0	1	1	1	1	3	4	0	0	5	8	2	4	0	0	0	1	0	0	0	1	1	2	2	F	F
1	1	1	3	1	0	3	3	1	4	2	0	0	4	0	0	0	0	0	1	11	0	0	0	1	1	0	0	0	0	1	1	T	F
1	0	0	0	0	1	1	1	2	0	3	1	1	6	1	1	1	0	1	3	1	1	2	2	0	1	1	0	2	0	3	3	F	F
2	2	3	1	3	1	2	0	1	2	1	0	4	0	1	5	3	1	0	0	0	0	0	0	0	0	1	1	1	1	2	2	F	F
2	2	0	2	1	0	0	0	0	0	3	1	0	1	3	1	5	0	2	4	1	0	0	1	1	1	1	0	2	0	3	3	F	F
0	2	0	1	1	0	0	0	3	1	1	0	0	0	0	0	0	6	3	5	0	4	8	2	1	1	0	0	0	0	0	1	F	F
1	0	0	0	0	0	1	1	1	2	3	3	3	0	2	0	0	7	1	0	0	1	5	0	1	0	3	1	2	0	2	0	F	F
1	1	1	0	2	1	3	3	0	2	0	0	0	1	4	1	2	1	1	2	1	1	0	0	1	1	1	1	1	2	2	3	F	F
0	3	0	3	1	0	1	3	1	3	1	3	1	9	3	1	0	4	0	0	1	0	0	0	0	0	0	0	0	0	1	1	F	F
0	2	3	1	0	1	3	1	1	1	0	4	6	1	3	0	0	5	0	2	2	1	0	0	0	0	0	0	0	0	1	2	F	F
2	1	1	1	0	0	0	0	0	0	0	8	1	1	0	0	0	2	1	0	10	1	6	0	1	1	1	0	0	0	1	1	T	T
0	0	1	1	1	1	1	1	0	10	0	1	1	3	1	1	1	4	3	4	3	0	1	1	0	0	0	0	0	0	0	0	F	F
1	0	4	1	2	1	0	2	0	0	0	6	1	1	0	0	1	1	0	0	10	2	0	0	1	1	0	1	1	2	0	1	T	T
1	0	1	3	2	0	2	1	1	2	3	1	2	4	1	0	0	1	0	0	2	0	5	0	1	1	1	1	1	1	1	1	F	F
0	1	2	0	2	0	2	0	0	5	7	3	1	2	0	1	0	0	0	0	0	0	4	1	2	2	1	1	1	1	1	0	F	F
1	1	2	0	2	0	2	0	3	2	1	0	0	4	0	1	0	4	0	4	5	1	0	2	0	0	0	0	0	1	2	2	F	F
0	0	0	0	0	0	0	0	0	0	0	13	2	1	1	1	1	3	0	1	0	4	4	1	2	0	2	0	2	0	1	1	F	T
1	0	0	0	0	0	1	1	1	2	3	1	6	0	1	1	0	4	1	3	0	4	0	1	1	1	1	0	1	1	2	2	F	F
2	2	1	0	0	0	0	0	0	1	7	13	0	0	0	1	0	5	1	0	0	4	1	0	0	0	0	0	0	0	1	1	F	T
1	1	0	0	0	0	0	0	1	2	1	3	1	2	1	0	1	8	2	0	11	0	0	0	0	0	0	1	1	2	0	1	T	F
2	1	0	0	0	0	0	0	2	1	2	7	1	0	0	0	0	1	8	0	5	2	3	2	0	0	0	0	0	1	2	0	F	F
0	2	0	2	1	0	0	0	2	0	0	11	0	2	1	1	0	0	0	1	12	0	1	1	0	0	0	0	0	1	1	1	T	T
2	0	1	1	0	0	0	0	1	0	0	10	3	0	0	1	1	3	4	1	0	0	3	1	1	1	2	0	1	0	2	1	F	T
1	1	0	0	0	0	0	1	7	1	8	0	3	5	3	0	0	0	0	0	2	0	5	0	0	0	0	0	0	0	1	2	F	F
0	2	1	1	0	0	0	0	1	0	0	9	0	2	0	0	0	4	1	0	11	0	2	1	0	0	1	1	0	2	1	0	T	T
1	0	0	0	0	0	0	0	2	4	1	0	1	1	2	1	1	0	1	3	13	1	1	1	1	1	1	1	1	0	0	1	T	F
0	0	0	0	0	0	0	0	1	1	4	0	0	5	0	1	1	1	0	4	13	0	3	0	1	1	0	0	1	1	0	2	T	F
0	2	1	0	0	0	1	1	0	0	3	12	4	1	1	0	2	3	0	1	0	2	4	0	0	0	0	0	0	1	1	0	F	T
1	1	0	0	0	0	0	1	0	6	1	1	1	1	2	0	1	0	2	3	0	5	6	0	1	1	1	1	0	1	1	2	F	F
2	2	1	1	1	1	0	0	0	7	1	1	0	0	0	0	0	0	1	0	11	2	1	1	1	1	1	1	0	1	2	0	T	F
1	1	0	0	0	0	0	0	0	1	1	14	0	0	0	0	0	8	1	0	8	1	0	2	1	0	0	0	0	0	0	1	T	T

2	0	1	1	0	0	0	0	0	0	0	0	1	1	3	0	1	4	0	0	12	3	1	0	1	2	0	3	1	2	1	0	T	F
0	0	0	0	0	1	1	1	0	4	4	10	1	7	3	1	1	1	1	0	2	0	0	0	0	0	0	0	0	0	1	1	F	T
2	1	2	1	1	0	0	1	0	3	4	1	7	1	0	0	0	1	1	1	1	2	0	3	2	1	1	0	0	1	2	0	F	F
1	1	0	0	0	0	1	1	1	5	3	1	2	0	1	0	0	0	0	0	2	2	7	1	2	0	2	0	2	1	4	0	F	F
0	0	0	0	1	1	1	1	0	0	0	1	2	1	5	0	0	10	6	2	1	0	0	0	1	1	1	1	1	1	1	1	F	F
4	3	1	1	1	0	1	1	2	0	0	2	5	5	0	0	0	0	0	0	0	0	0	1	0	1	1	0	2	2	3	4	F	F
0	0	0	0	1	1	1	1	2	3	1	1	3	4	2	1	1	1	4	3	0	1	1	1	1	0	0	0	1	2	3	0	F	F
1	3	2	2	1	1	1	1	2	3	1	0	1	4	1	2	0	2	4	0	0	0	0	1	0	0	0	1	1	0	2	3	F	F

References

[1] J. Baxter, A. Trigdell, L. Weaver, KNIGHTCAP: a chess program that learns by combining TD(λ) with game-tree search, in: Proc. 15th Internat. Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, 1998, pp. 28–36.

[2] D. Carmel, S. Markovitch, Learning models of opponent's strategies in game playing, in: Proc. AAAI Fall Symp. on Games: Planning and Learning, Raleigh, NC, 1993, pp. 140–147.

[3] D. Carmel, S. Markovitch, Pruning algorithms for multi-model adversary search, Artificial Intelligence 99 (2) (1998) 325–355.

[4] J. Christensen, R.E. Korf, A unified theory of heuristic evaluation functions and its application to learning, in: Proc. AAAI-86, Portland, OR, 1986, pp. 148–152.

[5] H.H.L.M. Donkers, Nosce Hostem: searching with opponent models, Ph.D. Thesis, Universiteit Maastricht, Maastricht, The Netherlands, 2003.

[6] H.H.L.M. Donkers, J.W.H.M. Uiterwijk, Programmming Bao, in: J.W.H.M. Uiterwijk (Ed.), Seventh Computer Olympiad: Computer-Games Workshop Proceedings, Technical Report CS 02-03, Universiteit Maastricht, Maastricht, The Netherlands, 2002.

[7] H.H.L.M. Donkers, J.W.H.M. Uiterwijk, H.J. van den Herik, Probabilistic Opponent-Model search, Inform. Sci. 135 (2001) 123–149.

[8] H.H.L.M. Donkers, J.W.H.M. Uiterwijk, H.J. van den Herik, Admissibility in Opponent-Model search, Inform. Sci. 154 (3–4) (2003) 195–202.

[9] H.H.L.M. Donkers, J.W.H.M. Uiterwijk, A.J. de Voogt, Mancala games—topics in artificial intelligence and mathematics, in: J. Retschitzki, R. Haddad-Zubel (Eds.), Step by Step, Proc. fourth Colloq. on 'Board Games in Academia', Fribourg, Switzerland, 2002(editions Universitaires).

[10] H.J. van den Herik, J.W.H.M. Uiterwijk, J. van Rijswijck, Games solved: now and in the future, Artificial Intelligence 134 (1–2) (2002) 277–311.

[11] J.H. Holland, Adaption in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.

[12] H. Iida, J.W.H.M. Uiterwijk, H.J. van den Herik, Opponent-Model search, Technical Report CS 93-03, Universiteit Maastricht, Maastricht, The Netherlands, 1993.

[13] H.J.R. Murray, A History of Board Games other than Chess, Oxford University Press, Oxford, UK, 1952.

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

[15] L. Russ, The Complete Mancala Games Book, Marlow & Company, New York, 2000.

[16] A.L. Samuel, Some studies in machine learning using the game of Checkers, IBM J. Res. Develop. 3 (1959) 210–229.

[17] A.L. Samuel, Some studies in machine learning using the game of Checkers II—recent progress, IBM J. Res. Develop. 11 (1967) 601–617.

[18] C. Spearman, The proof and measurement of association between two things, Amer. J. Psychology 14 (1904) 72–101.

[19] A.J. de Voogt, Limits of the mind, towards a characterisation of Bao mastership, Ph.D. Thesis, University of Leiden, The Netherlands, 1995.

[20] T. Yoshioka, S. Ishii, M. Ito, Strategy acquisition for the game Othello based on reinforcement learning, IEICE Trans. Inform. Systems E82-D(12) (1999) 1618–1626.