# NOSCE HOSTEM

SEARCHING WITH
OPPONENT MODELS

HHLM DONKERS

# NOSCE HOSTEM
Searching with Opponent Models

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit Maastricht,
op gezag van de Rector Magnificus,
Prof. dr. A.C. Nieuwenhuijzen Kruseman,
volgens het besluit van het College van Decanen,
in het openbaar te verdedigen
op vrijdag 5 december 2003 om 12.00 uur

door

Hiëronymus Hendrikus Lambertus Maria Donkers

SIKS Dissertation Series No. 2003-13
*The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.*

ISBN 90-5278-390-X
Universitaire Pers Maastricht

Printed by Datawyse b.v., Maastricht, The Netherlands.

# Preface

During the time I was a programmer and system manager at the department of Medical Informatics, I saw many Ph.D. students doing their work. From a distance, such Ph.D. research seemed not that difficult, so I decided to follow this track. After re-entering and completing my academic studies, I was able to begin with my own Ph.D. research; first on the subject of my Master's thesis: reasoning with uncertainty. Later I switched to the subject of computer game-playing, which was a new challenging topic for me at that time. My motivation for the new topic was almost automatic since I like games and playing games, moreover, IKAT had many people who knew much of the subject and who could criticise me. Apparently, an issue which encouraged me to do even better. Submitted papers were accepted and it finally resulted in this thesis.

Research on computers and board games is a joy for me since because, among other things, it is fundamental science, revealing the properties of complex structures that are hidden inside games. Owing to my relatively late start I was able to observe the subject at a somewhat more abstract level. This sometimes led to interesting discussions in the games group. The goal of my research was to investigate the properties of a promising search method, Opponent-Model search, that had been invented six years earlier. At the start, the majority of my research was more theoretical than practical. As time went on, however, I became involved in competitive computer game-playing and a small olympic flame was lit inside me.

I am convinced that the theoretical and practical results obtained during this research will be of great value to future researchers. However, the main outcome of this research is to be summarized as: whatever the promises of Opponent-Model search, its applicability is clearly less than expected previously.

The enormous efforts, big support and tough lessons by Jaap van den Herik, made it possible for me to complete the research successfully and to submit this thesis. Jos Uiterwijk kept me on the road and was never tired to correct the same typos in whatever draft version again and again. I wish to express my gratitude to IKAT and the Faculty of General Science for the opportunity to perform and fulfil this research task. I thank all my colleagues for their patience and co-operation and for taking the extra work that came on their shoulders. In particular I would like to thank Floris Wiesman, who introduced me to the wonders of LaTeX and who was always willing to answer those silly little questions on how to write a thesis. My fellow students taught me much about computer game-playing, so I thank Mark Winands, Levente Kocsis, and Erik van der Werf for their inspiring discussions and supportive

comments on my research. I thank Sjoerd Druiven for our discussions on game theory. Eric Postma and Ida Sprinkhuizen-Kuyper are recognized for their advice on the machine-learning issues. Additionally I would like to acknowledge Marlies van der Mee for her administrative support in the final stage and Joke Hellemons for her continuous request for 'lijstjes'.

With much pleasure I recall the discussions with Don Beal, Ken Chen, Hiroyuki Iida, and Guy Haworth. Alex de Voogt not only introduced me to the intricacies of the game of bao and other mancala games, but he also made me enthusiast to play these games myself. I am grateful for our co-operation. Jean Retschitzki is acknowledged for sending me copies of some historical papers on kalah.

Doing a Ph.D.-research project part-time means that you have to split your attention over different subjects and to preserve a delicate balance between research and other activities. It also means that many evenings and weekends are occupied. Therefore I apologise to my friends and relatives for often not being around. My parents deserve a special word of thanks since they have always supported me, even in my headstrong choices. Finally, I wish to thank Hans for his love, caring, and patience during all these years.

Jeroen Donkers, 2003

*"Nosce Hostem"*

*Motto of the 532d US Military Intelligence Battalion (Stuttgart)*

# Contents

## List of Algorithms

## List of Figures

## List of Tables

# Chapter 1

# Introduction

*Artificial Intelligence (AI) is an interesting research area with challenging problems. One of the problems is the game of chess. Already at the start of AI research (around 1950) it was on the list of problems to be "solved". Here solving means building a program that could play at a par with or even stronger than the human world champion. In 1997 the chess system* DEEP BLUE *(Hsu, 2002) surprised the non-academic world by winning a match against the world champion Gary Kasparov. Some then believed that this victory was the end of chess as a research area, others argued that the game still was not solved and that many mysteries remained to be unravelled. For instance, the complex method of using opponent models in search was in the mid 1990s still in its infancy. The fact that* DEEP BLUE *team profited considerably from the co-operation with grandmasters who studied Kasparov's games and fed the system with knowledge on his style, can be considered as its actual incorporation in match play. The general goal of this thesis is to investigate to which extent knowledge of the opponent can be used to improve computer game-playing.*

---

**Chapter contents:** Section 1.1 gives an introduction to computer game-playing. Section 1.2 is dedicated to mathematical game theory. It also introduces the notational system. Section 1.3 deals with heuristic search in computer game-playing. The concept of Opponent-Model search is introduced in section 1.4. The formulation of our problem statement and research questions is described in section 1.5. Our methodology is given in section 1.6. Finally, section 1.7 provides an overview of the remainder of the thesis.

## 1.1 Computer Game-playing

In 1950, Claude Shannon wrote in his seminal paper on programming a computer for playing chess: "chess is generally considered to require 'thinking' for skilful play; a solution of this problem will force us either to admit the possibility of mechanized thinking or to further restrict our concept of 'thinking'." According to Van den Herik (1983), this statement can be seen as the birth of artificial-intelligence research.

In the fifty years thereafter, research in computer game-playing[1] has seen many successes. Nowadays, computers are able to play chess at world-champion level and in other games (e.g., checkers, othello, and backgammon) computers had similar successes. Moreover, a number of games have even been solved by computers (e.g., connect-four, nine-men's-morris, and awari). For an overview of the achievements of computer game-playing so far, we refer to Schaeffer and Van den Herik (2002).

Results achieved in one game, however, do not guarantee success in other games. For example, the game of go, although having far simpler rules than chess, is not played very well by computers because the successful techniques developed for chess do not apply satisfactorily to the game of go. This means that new techniques need to be developed in order to play this game at a similar competitive level as chess.

A particularity of research in computer game-playing is the emphasis on competition. Since computers play frequently in competitions with other computers and in matches against human players, there is a constant urge for improvement. This leads to subsequent refinements of the algorithms involved. A number of these refinements have found their way successfully to other areas of computing. A telling example is the application of transposition tables in single-agent search (see, for instance, Romein *et al.*, 1999). In this respect, computer game-playing is nowadays more like the *Formula One of AI* than the *Drosophila of AI*, as it was called by Alexander Kronrod in 1965 (see McCarthy, 1990; 1997). In his well-known book on the role of game playing in human culture, *Homo Ludens*, Huizinga (1938) hardly allowed any room for competition (*agon*) in modern science. He was unable to foresee such a close relation between game and science as in competitive computer game-playing. It is encouraging to observe that since half a decade, the competition-driven research is also applied in other fields of AI: the *RoboCup* is annually played by teams of AI researchers who test their theories of computer vision, agent technology, planning, and robotics.[2] Moreover, in conjunction with the ACM conference on Knowledge Discovery and Data Mining, an annual competition is conducted in data mining: the *KDD Cup*.[3]

A general theoretical question in research on computer game-playing is why computers play some games, like chess, successfully and other games, like go, not. Some of the answers might be found in the issues that Junghanns (1998) described concerning the basic algorithm of computer game-playing. One of them is that this basic algorithm, $\alpha$-$\beta$ search, does not explicitly use knowledge of the opponent. Anyone who plays board games will agree that it is natural to use knowledge of the opponent

[1]In this thesis we will study the playing of human games by computers, not the playing of computer games by humans. Hence we will write "computer game-playing" instead of "computer-game playing".

[2]See: http://www.robocup.org.

[3]See: http://www.kdnuggets.com/datasets/kddcup.html.

at some point in order to exploit the opponent's weaknesses. In this thesis we will study a number of algorithms that use an explicit model of the opponent. We study these algorithms both theoretically and experimentally.

In our theoretical research we will use the language of mathematical game theory because we feel that the distance developed between computer game-playing and mathematical game theory should be decreased. Some issues in computer game-playing have also been examined in mathematical game theory, but due to the separation of the scientific communities, the results of these studies are almost not referred to in the literature on computer game-playing. We mention one illustrative case in chapter 3. In passing we mention that the same holds for the opposite direction.

Below, we start with the mathematical background of game theory in order to establish the formal language of the thesis, then we return to computer game-playing, and finally we formulate our problem statement and approach precisely.

## 1.2 Game Theory

In this section we briefly introduce mathematical game theory and its relation to computer game-playing. Subsection 1.2.1 informally introduces the type of games that are of interest for our research. A concise formal base of relevant issues is provided in subsection 1.2.2 and the fundamental notion of game trees is discussed in subsection 1.2.3. To clarify the theoretical definitions, subsection 1.2.4 provides an elaborated example. Subsection 1.2.5 links game theory to the practice of game-playing by computers.

### 1.2.1 Games of interest

In this thesis we study (board) games like chess and checkers. In mathematical game theory, these games are called *finite two-player zero-sum games with perfect information* (Fudenberg and Tirole, 1991).

We restrict ourselves to games for *two* players because their properties are quite different from games for one player (puzzles, e.g., Sokoban) and for three or more players (e.g., poker). Consequently, the results for two-player games are hardly applicable in puzzles and multi-player games, and vice versa.

The games of our interest are *zero-sum games*: the outcome for a particular game for one player is the negative of the outcome of the other player. If one player wins a certain amount, then the other player loses exactly the same amount.

The games are characterized by *perfect information*. This means that both players have access to all information needed to know the state of the game at any moment.[4] There are no such things as hidden cards, or an invisible stock. Furthermore, there is no chance involved (no dice, no blind dealing of cards). Perfect information does not mean that all information is actually *visible*. For instance,

[4]The meaning of the term 'perfect information' has evolved over time. In Von Neumann and Morgenstern (1944), another definition was used, close to the modern definition of 'complete information'. In this thesis we adopt the widely accepted game-theoretic notions as formulated in Fudenberg and Tirole (1991).

many games (chess, go) have a rule that forbids (unlimited) repetition of positions. Such a rule implies that the history (the positions that appeared earlier in that game) is part of the state of a game. The configuration of pieces on a chess board does not reveal previous positions, but both players are assumed to remember the moves that have been made and in this way they can reconstruct the history perfectly.

The games that we study must be *finite*: the number of possible moves at every position must be finite and the duration of a game must be finite.

In game theory, finite two-player zero-sum games with perfect information are the most elementary type of games. They might be regarded as trivial since the solution of such game is easily defined. Such a solution consists of the following algorithmic strategy for both players: in every position, select the move that maximizes the payoff for the player to move, thereby assuming that the same strategy is followed for all possible positions in the game. This is called the Maximin Criterion and leads to the Minimax algorithm (Von Neumann, 1928). Although this solution is easily *defined*, it is not always easily *found*, especially in the case of large, complex games like chess. The latter is the domain of computer game-playing.

### 1.2.2 A concise formal base

After this informal introduction, we will now present a concise formal base of relevant issues in game theory. The goal of this section is to introduce the notations and to define the game-theoretic terms that are used throughout the thesis. For a detailed formalization of game theory, we refer to the textbook by Fudenberg and Tirole (1991).

The word "game" will be used in two senses: first, as a set of rules (e.g., the game of chess), second, as the activity of playing according to these rules (e.g., a game of chess). Also the common term "move" has two meanings: first, as a possible action during the course of a game (e.g., Queen to a5), second, as indication of an action by the first player and the response by the second player (e.g., "the first move of the game was: 1. e4 c5"). In the last notion, the term *ply* is used to indicate "half a move", an action of one player only.

The class of games described informally above can be specified formally as follows: A two-player zero-sum game with perfect information, $G$, is a tuple $G = (M, H, E, p, m, V)$. In this tuple, $M$ is the set of all possible *moves* in the game, which can include passes and resignations.

$H$ is the set of all legal *move histories* (ordered lists of moves $\langle m_1, m_2, ..., m_n\rangle$, $m_i \in M$) that can occur in the game, starting from the beginning. $H$ includes the empty history $\langle\rangle$. The length $|h|$ of a history is the number of moves it contains ($n$). For every game it must hold that if $h = \langle m_1, m_2, ..., m_n\rangle \in H$ then all subhistories $h(k) = \langle m_1, m_2, ..., m_k\rangle, k < n$ of $h$ are also element of $H$. A game is defined to be finite if and only if $H$ is finite.

The set $E \subseteq H$ consists of all move histories that result in a legal ending of the game. The positions reached after the last move of those histories are the *terminal positions*.

The *player function* $p : H \rightarrow \{1, 2\}$ denotes the player to move after every history. By definition, $p(\langle\rangle) = 1$. For games with alternating players $p(h) = 1$ if $|h|$ is even and $p(h) = 2$ if $|h|$ is odd.

The *move function* $m : H \to 2^M$ denotes the set of legal moves after each history in $H$. For $m$ must hold that if $h \in H \setminus E$ then $m(h) \neq \varnothing$ and if $h \in E$ then $m(h) = \varnothing$. Furthermore, $h$ concatenated with any member $m$ of $m(h)$ must be in $H$ for all $h \in H$. (We use the notation $h + m$ to denote the concatenation of history $h$ with move $m$.)

The *value function* $V : E \to \mathbb{R}$ gives the score (payoff) of the game for every history in $E$. The scores for the players are $V_1(h) = V(h)$ and $V_2(h) = -V(h)$.

A player $p$'s *strategy* for game $G$ is a function $s_p : H \to M$ that picks a particular move for every history after which the player $p$ is to move. Of course, $s_p(h) \in m(h)$ if $p(h) = p$, else $s_p(h)$ is undefined. If two players each follow a particular strategy, then the result of that pair of strategies $\mathbf{s} = \langle s_1, s_2 \rangle$ is a unique history $h(\mathbf{s})$ and a corresponding strategy score $v(\mathbf{s}) \equiv V(h(\mathbf{s}))$. History $h(\mathbf{s})$ is uniquely determined by the pair $\mathbf{s}$, because for every $k$, the move following subhistory $h(\mathbf{s})(k-1)$ is given by $s_p(h(\mathbf{s})(k-1))$, where $p$ is the player to move.

A *solution* $\mathbf{s}^* = \langle s_1^*, s_2^* \rangle$ of a game is a pair of strategies such that there is no pair $\mathbf{s}' = \langle s_1^*, s_2' \rangle$ with a score *lower* than $v(\mathbf{s}^*)$ and there is no pair $\mathbf{s}' = \langle s_1', s_2^* \rangle$ with a score *higher* than $v(\mathbf{s}^*)$. All solutions of a particular game necessarily have the same score (due to the Minimax Theorem for perfect-information games, see Von Neumann, 1928), so it is justified to define the score $v(G)$ of a game as the score of one of its solutions. This score $v(G)$ is also called the *game-theoretic value* of the game. History $h(\mathbf{s}^*)$ induced by the solution $\mathbf{s}^*$ is a *principal variation* of the game.

A *subgame* $G(h)$ of a game $G$ is the restriction of game $G$ to all histories that share subhistory $h \in H$ (including $h$ itself). The game-theoretic value $v(G(h))$ of this subgame is also referred to as the value of $h$ itself: $v(h)$. For all $h \in E$, the subgame $G(h)$ is trivial, because there is no move left. For the empty history $\langle\rangle$, the subgame $G(\langle\rangle)$ is equal to $G$ itself.

This concludes the formal notations from game theory. We will use these notations to express our ideas and results. Since the work by Von Neumann and Morgenstern (1944), the languages of mathematical game theory and computer game-playing have diverged. We attempt to bridge the gap by adopting these notations; they allow us to compare ideas and utilize results from both areas.

### 1.2.3 Game trees

In computer game-playing literature, games are usually represented in a graphical form, the *game tree* (see figure 1.2 for an example). In a game tree, the arcs are marked with moves from $M$ in such a way that every path in the tree starting from the root corresponds to a history in $H$ and that every history in $H$ occurs as a path from the root. The nodes in a game tree correspond to game states. The set $E$ corresponds to all paths leading from the root to the leaf nodes of the game tree. The internal nodes of the tree are labelled (through their shape) by the player function of the history corresponding to the path leading to that node: the starting player is indicated by a square, the second player by a round node. Leaf nodes are labelled by the score of the corresponding histories. Subgames correspond to subtrees in the game tree. The *value* of any node in the game tree is the game-theoretic value of the corresponding subgame.

The game tree can be used to give an alternative formulation of the optimal strategy (or solution) for both players. A strategy for a player consists of the selection of a branch for every node in the tree at which the player is to move (the resulting subtree is called a *solution tree*, see De Bruin, Pijls, and Plaat, 1994). For leaf nodes, the optimal strategy is trivial because nothing is to be selected and its value is given by the value function of the game. For internal nodes the optimal strategy for each player is as follows: first determine (by recursion) the value of all direct child nodes; then select the child node depending on who is to move; if the first player is to move, select a child node with the highest value, otherwise select a child node with the lowest value. The value of the node itself is equal to the value of the selected child node. This solution is called the Minimax solution and the procedure to determine the solution is called Minimax search. The value of this solution is therefore also referred to as the *Minimax value* of the game.

### 1.2.4 An example game

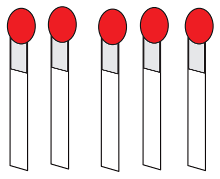


Figure 1.1: The game nim-5 is played with 5 matches.

A small example game that might serve to explain the concepts in this chapter is nim-5 (figure 1.1). This is a simple variant of the well-known game of nim. On a table are five matches. Two players remove on turn one, two, or three matches. The player that removes the last match wins the game. Clearly, nim-5 satisfies all our criteria for a zero-sum two-player game with perfect information.

In the game of nim-5, the set of moves is $M = \{1, 2, 3\}$ and the complete set of histories in this game is $H = \{\langle\rangle$, 1, 2, 3, 11, 12, 13, 21, 22, 23, 31, 32, 111, 112, 113, 121, 122, 131, 211, 212, 221, 311, 1111, 1112, 1121, 1211, 2111, 11111$\}$[5]. These are all the sequences of the numbers 1, 2 and 3 of which the total of the numbers does not exceed 5. The terminal histories are: $E = \{$23, 32, 113, 122, 131, 212, 221, 311, 1112, 1121, 1211, 2111, 11111$\}$, being all sequences that add exactly to 5. Because players alternate their turns, $p(h) = 1$ if $|h|$ is even and $p(h) = 2$ if $|h|$ is odd. The move function $m$ for nim-5 includes all moves for a given history $h$ that do not take more matches than there are still on the table. Finally, the score for even-length histories in $E$ is $-1$ and for odd-length histories it is 1.

The game tree for nim-5 is depicted in figure 1.2, whereas figure 1.3 gives the optimal strategies for the game tree of nim-5. Player 1 can win the game if she[6]

[5] We use the abbreviation $i$ for the sequence $\langle i\rangle$, $ij$ for the sequence $\langle i, j\rangle$, *et cetera*.

[6] We use female gender for the first player and male for the second, when appropriate.

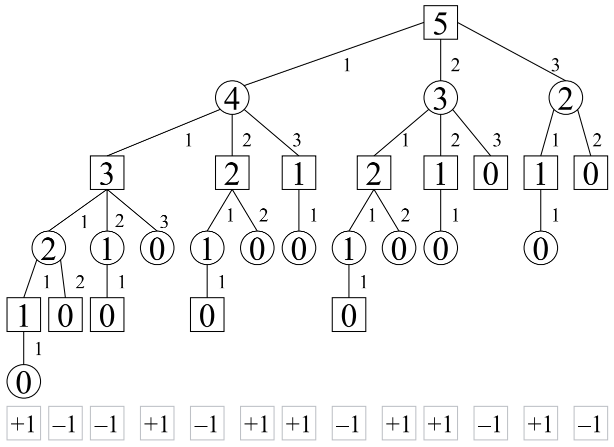


Figure 1.2: Game tree for nim-5. The square nodes represent positions in which player 1 is to move, the round nodes those with player 2 to move. Inside the nodes are the number of matches still on the table. Below the tree are the payoffs for every complete history.

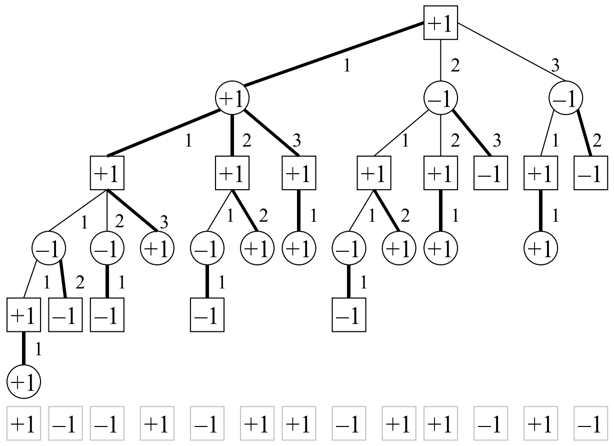


Figure 1.3: Optimal strategies for nim-5. Inside the nodes are the values of the subgames and the thick lines indicate the optimal moves for both players.

starts with picking one match. Observe that after player 1 has picked a single match, all possible moves by player 2 are optimal (equally bad), resulting in three different optimal strategies for player 2.

### 1.2.5 Game-playing

In order to play a game perfectly (that is, using an optimal strategy), it is not necessary to determine a complete solution that specifies a move for all nodes in the tree where the player is to move. The task of game-playing is just to provide the best move, according to the optimal strategy $s^*(h)$ when a history $h$ of moves is already played. This task is equivalent to determining the *Minimax value* of the subgame $G(h)$.

In research on computer game-playing, many algorithms have been developed that determine the Minimax value of a game tree without determining a complete solution. Some examples are $\alpha$-$\beta$ search (Knuth and Moore, 1975), SSS* search (Stockman, 1979), Conspiracy-Number search (McAllester, 1988), Proof-Number search (Allis, Van der Meulen, and Van den Herik, 1994), and MTD(f) (Plaat, 1996). For an overview, see Junghanns (1998).

## 1.3 Heuristic Search

Game trees for board games are usually large. This makes it infeasible for humans and computers to determine optimal moves, except for the last moves of a game when the subgame that has to be solved becomes sufficiently small. The impossibility of finding a game-theoretic solution for chess, checkers, or go has stimulated the development of a range of heuristic methods to replace the optimal strategy. In subsection 1.3.1 a method is introduced that tries to find the Minimax value of a reduced game tree. Subsection 1.3.2 discusses a number of alternative, non-Minimax methods.

### 1.3.1 Minimax search

The most widely used heuristic-search technique is to determine the best move according to the optimal strategy for a *reduced game* and use the result as an approximation of the optimal strategy for the complete game. Game $G' = (M, H', E', p, m', V')$ is a reduced game of game $G = (M, H, E, p, m, V)$ if (1) $H' \subseteq H$, (2) $E' \supset (E \cap H')$ and (3) $m'(h) \subseteq m(h)$. The first condition imposes that the reduced game does not allow more moves than the original game. The second condition implies a reduction of the length of some histories: there are histories that are considered complete in the reduced game but are not complete in the original game. For any 'truncated' history $h \in E' \setminus E$, the function $V'$ returns a provisional payoff that replaces the game-theoretic value of that history's subgame in the original game $G$. This function is normally called the *heuristic static evaluation function*. For truly complete histories, $V'$ is kept equal to $V$, except for a scaling factor that makes it possible to discriminate between terminal positions of the original games and heuristic values of truncated histories. The third condition means that for certain histories, only some selection of moves of the original game may be considered in the reduced game.

Figure 1.4 shows a reduced game for nim-5, where $H'$ is reduced to all histories of maximum length 2 and $E'$ contains all histories of exactly length 2. The following

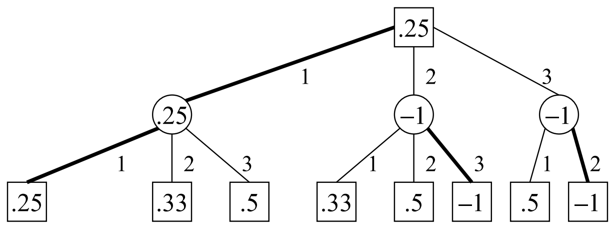


Figure 1.4: A reduced game for nim-5. Heuristic evaluation values are given within the leaf nodes.

heuristic evaluation function is used for truncated histories: $V'(h) = 1/(6 - n(h))$, where $n(h)$ is the number of matches already picked up in that history. As the figure shows, the optimal strategy of this reduced game leads to the same move (picking a single match) as the optimal strategy of the complete game. This indicates that the heuristic evaluation function is of such a quality that it leads to a correct prediction of the optimal move on the basis of truncated histories.

In computer game-playing, a reduced game is called a *search tree* and the maximum length of the truncated histories is the *search depth.* Algorithms, such as $\alpha$-$\beta$ search, that determine the score of a reduced game are normally called (Minimax) *game-tree search algorithms.* The actual size of the search tree depends on the available resources such as: memory, time, processor speed, and the number of processors. It also depends on the ability of the search algorithm to prune parts of the search tree. This *pruning* of the search tree is of major importance to practical computer game-playing because the more can be pruned, the larger (deeper and wider) the search tree can be. We want to distinguish between the *search tree*, which is the tree that fully represents the reduced game, and the *visited tree* which is a subtree of the search tree containing only nodes that are investigated by the search algorithm. Pruning a branch in the search tree then means that the branch is not visited and thus is not included in the visited tree. Search algorithms like $\alpha$-$\beta$ search can prune significantly more if the search tree is ordered in a specific way. Therefore, much effort is put into achieving a good tree ordering (see Marsland (1983)).

During an actual game, a game-playing computer program starts at every turn a new game-tree search for a reduced game of the subgame determined by the moves that have already been played. By convention, the root node of a search tree is always a maximizing node, regardless of the player who is to move. If this happens to be player 2, then the scores of the game are negated $V() \leftarrow -V()$ and player 2 now maximizes whereas player 1 minimizes. Leaf nodes of the search tree are called *terminal nodes* if they correspond with terminal positions. The depth of nodes in a search tree is expressed in *plies.* The root node is on ply 0 by convention, its children are on ply 1, *et cetera.*

### 1.3.2 Alternatives for Minimax search

Although it appears that Minimax search is a sufficiently good approach for computer chess in order to play at world-champion level, this success was mainly possible owing to a high efficiency, a good heuristic evaluation function, efficient pruning, good opening books, and a set of endgame databases. The same approach, however, proved to be not so successful in a number of other games, for instance, go and shogi.

It is not guaranteed that solving a sequence of reduced games during the course of a game is the best way to defeat the opponent. There are two main reasons.

The first reason is that the evaluation function is often not a good approximation of the game-theoretic value. This means that the strategy resulting from Minimax search can be far away from the optimal strategy. We return to the issue of evaluation functions in chapter 4.

The second reason is that the goal of game-playing is not to find an optimal strategy, but to *win the game* against a human being or a computer opponent. Minimax search assumes that the opponent of the player is using the same evaluation function and search method as the player itself. It is not likely that this is ever the case (except in endgame situations). The task of game-playing thus becomes to find the best strategy against a given opponent. This need not be an optimal strategy for the game, in the game-theoretical sense.

Because in most positions we are not able to determine an optimal strategy, it is obvious that alternatives for Minimax search should be investigated in order to approximate the optimal strategy more adequately. Junghanns (1998) discusses a number of alternatives for $\alpha$-$\beta$ search. Some of these alternatives are still Minimax search-like methods, but Junghanns also describes several attempts to establish adequate strategies for a game in fundamentally different ways.

Junghanns' (1998) most important observation is that the search tree contains a large amount of information that could be used for establishing such an adequate strategy. However, the information is disregarded by Minimax search because this search method only needs to compute the game-theoretic value of the root node. Some of the alternative search methods attempt to use more of the information available in the tree. For instance, they explicitely use the number of "good moves" in a certain position. One of these methods is M & N search (Slagle and Dixon, 1970) which we will analyse in section 2.1.2.

## 1.4 Opponent-Model Search

A specific alternative for Minimax search that explicitly takes the opponent into account is *Opponent-Model search* (OM search). This search method was developed by Iida *et al.* (1993b) and simultaneously by Carmel and Markovitch (1993). The method assumes that a player has complete knowledge of the opponent's strategy. The opponent is assumed to use Minimax search and at his turn to be unaware of the player's strategy. Given this knowledge, the search method determines the best response for the player with respect to the opponent's strategy. The advantage of having such an opponent model is adequately illustrated below, again using the game of nim-5. Suppose that player 1 will never take three matches and that player 2 knows this. In figure 1.5 a representation is given for this strategy and its consequences. The knowledge reduces the strategy of player 1, and consequently player 2 can easily find an answer to this strategy that allows him to win. A principal variation reads:

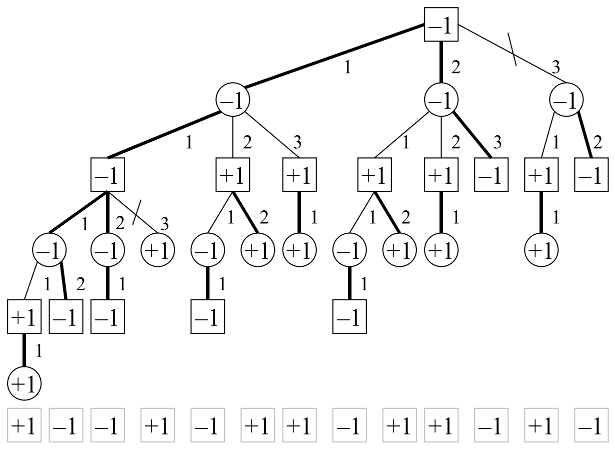


Figure 1.5: Optimal strategies for both players assuming player 1 never picks three matches (these choices have been struck out). If player 2 knows this strategy (opponent model), then he can force a win.

1,1,1,2. On the basis of this small example it is tempting to expect that using knowledge of the opponent can improve computer game-playing.

In practice, however, complete knowledge on the opponent is almost never available, but some uncertain knowledge might be available. Therefore, we developed an extension of OM search in which the opponent's strategy is *approximated* by a probabilistic mix of a number of known strategies (that is, at every move, the opponent uses a dice to determine which pure strategy to use for the next move). This search method is called *Probabilistic Opponent-Model search* (PrOM search). The mix of strategies allows for adaptation to a real opponent.

The problem with OM search (and also with PrOM search) is that it is unknown whether these alternative search methods are indeed able to fulfil the expectation that they improve computer game-playing. Intuitively, it is clear that having knowledge of your opponent at your disposal should provide an advantage, but until now, nobody has used OM search or PrOM search in tournament programs. Hardly any experiments have been performed in realistic settings with these methods, so it is not known: (1) how large the advantage might be, (2) how much risk might be involved and (3) what the computational burden might be of these search methods. These three questions will lead us to the problem statement of this thesis.

## 1.5 Problem Statement and Research Questions

As stated above, the context of our research is computer game-playing. We observed that although computer game-playing has seen many successes, there is still much unknown on the nature of search algorithms, especially on the effect of using knowledge of the opponent as is done in OM search. The small example of the game nim-5 suggests that using such knowledge might be profitable. However, playing actual games is much more complex, and therefore we should first perform theoretical investigations and then combine them with the examination of the results of practical algorithms. The observations and considerations above have led us to the following problem statement.

> **Problem statement**: *Under what conditions can Opponent-Model search and Probabilistic Opponent-Model search improve computer game-playing?*

To answer this question, both search methods have to be studied in detail in order to learn their properties. As a guideline to our research we formulated four explicit research questions. They deal with (1) theoretical properties, (2) efficient implementations, (3) the nature of the game positions in which OM search or PrOM search is efficient, and (4) the challenge how to learn and formulate an opponent model.

> **Research question 1**: *What are the important theoretical properties of Opponent-Model search and probabilistic Opponent-Model search?*

Because of the complex game trees involved, purely theoretical research will not be able to provide a full answer to the problem statement. We also need to find out whether the search methods have any practical use. Therefore, the methods have to be implemented into algorithms and the properties of those algorithms have to be investigated.

> **Research question 2**: *How can these search methods be implemented efficiently?*

If efficient implementations of the search methods are possible, these algorithms have to be tested in realistic settings because the ultimate reply to our problem statement depends on whether and when these algorithms work in practice.

> **Research question 3**: *What is the nature of the practical circumstances in which OM search and PrOM search perform better than the current implementations of Minimax search?*

Since Probabilistic Opponent-Model search theoretically has the possibility to be used when learning opponent models, we wish to test whether this is possible in a practical setting.

> **Research question 4**: *Can probabilistic Opponent-Model search be used in practice to learn an opponent model?*

If this fourth research question is answered positively, PrOM search could be applied to learn, for instance, the strengths and weaknesses of a chess pupil, or to build a game-playing program that adapts itself during a game to the opponents' style. The answers to these four research questions will allow us to formulate a reply to the problem statement above.

## 1.6 Methodology

To answer the four research questions above, we apply both theoretical and empirical methods. The theoretical research will reveal the general properties of OM search and PrOM search and will also lead to the implementation of these methods into several algorithms. Moreover, a theoretical analysis of the algorithms will be given. With respect to the power of generalization, theoretical results are frequently preferred over empirical results. However, due to the high complexity of the game-tree search algorithms and due to the particular shape of the game trees in real games, theoretical research alone is not sufficient to provide adequate answers relevant to the research questions formulated above.

Next to the theoretical approach, we employ a more empirical approach. Here we study the behaviour of the developed search algorithms on randomly constructed game trees to measure the average case of the computational complexity. In the case of learning opponent models, we use randomized methods to measure the properties of our theoretically developed learning method.

The above approach is still somewhat theoretical since general randomized trees are used. For a pure empirical comparison we submit our algorithms to three actual games. They are: lines of action (LOA), chess, and bao. It means that we embed the algorithms in state-of-the-art game engines and test them under relaxed tournament conditions: we vary the parameters in a controlled way and let two different engines play against each other many times. Owing to randomness in the engines or in the start positions, the experimental results will be rather diverse. However, it will allow for a statistical interpretation.

The theoretical and empirical approaches are intertwined: the theoretical hypotheses ask for experiments, and unexpected results ask for theoretical explanations. Yet, for clarity, we separate the descriptions of our theoretical and empirical research. Theoretical research is described in detail in the chapters 3, 4, and 6. The detailed empirical research methods are explained in chapters 5 and 7.

## 1.7 Thesis Overview

Chapter 2 starts with an overview of methods that preceded OM search. Thereafter the history and definition of OM search is provided, followed by an overview of variants of OM search.

In chapter 3, OM search is analysed together with its implementations. First, we describe some properties of the search method. Next, different implementations of OM search are presented and analyzed. In chapter 4, we define conditions for a possibly successful application of OM search. These chapters deal in particular with research questions 1 and 2 and serve as a stepping stone for the chapters 5 and 6.

Chapter 5 describes the experiments performed with OM search. First some experiments on random game trees are presented. Then experiments in the games of lines of action, chess, and bao are described. The chapter provides a variety of background information too, viz. on the methods used in the experiments and on the three games involved in the experiments. Research question 3 is the main lead of this chapter.

Chapter 6 discusses Probabilistic Opponent-Model search. After a short exposition of the method and its motivation, we give an overview of related research. This is followed by the derivation of theoretical properties of PrOM search, an algorithm for the method, and its analysis. The last part of this chapter describes several methods for learning a probabilistic opponent model. The chapter deals with the research question 1, 2, and 4 (see chapters 3 and 4).

In chapter 7, the experiments of PrOM search are described. Some of the experiments are performed in parallel to the experiments of chapter 5. Next to research question 3, this chapter also answers research question 4.

The results of the experiments of chapters 5 and 7 are compared and discussed in chapter 8. This chapter first answers the four research questions and then comes to a conclusive answer of the problem statement.

# Chapter 2

# Opponent-Model Search

*In 1993, two research groups, one in Israel and one in the Netherlands, simultaneously discovered an alternative for Minimax search that took knowledge of the opponent into account. They both called it: Opponent-Model (OM) search. The main difference with the Minimax approach is that the move selected at min nodes is not the worst possible move (or the best for the opponent), but the move that the opponent* would *select. This enables the exploitation of a weakness, spotted in the opponent's decision-making mechanism.*

---

**Chapter contents:** In section 2.1 we present how different researchers treated the use of opponent knowledge before OM search was developed. The birth of OM search is described in section 2.2. In section 2.3 this method is explained in terms of the inventors. Section 2.4 presents further developments in OM search. In section 2.5 we discuss some related topics in game theory. The chapter ends with a conclusion (section 2.6).

## 2.1 Foregoing Research

The idea of taking the opponent into account in game-tree search has been studied for almost twenty years and a number of solutions have been proposed before Opponent-Model search was developed. In this section we treat five approaches in chronological order followed by a comparison. They are: M & N search, *-Minimax search, Jansen's HP (Heuristic Program), Jansen's BO (Best Optimal), and Speculative Play. At the end of the section, we discuss how opponent knowledge is used in today's computer game tournaments.

### 2.1.1 Terminology

To describe the search methods adequately, we replace the range of terminology used in the literature referred to, by the terminology and notation provided in chapter 1. It facilitates the comparison between the different approaches that we would like to discuss.

In the sequel, we treat the notions 'history' and (game-tree) 'node' as equivalent and we use the symbol $h$ for histories and for nodes. We also use the abbreviation 'value of a move $m$' if we mean the subgame value $v(h+m)$. Since in our games, player 1 is trying to maximize the payoff and player 2 is trying to minimize it, we call them MAX and MIN, respectively. Sometimes we refer to MAX simply by 'the player' and to MIN by 'the opponent'. The nodes in a game tree where MAX is to play (viz., $p(h) = \text{MAX}$) are called max nodes, the nodes where MIN is to play (viz., $p(h) = \text{MIN}$) are called min nodes.

### 2.1.2 M & N search

The first attempt to incorporate an opponent model in game-tree search is the approach by Slagle and Dixon (1970). At the base of their M & N search method lies the observation that Minimax search does not make a distinction between nodes in which there are several moves with good values and nodes in which there is only one move with a good value. In order to compute the game-theoretic value of a complete game tree, the number of good moves in a node is indeed irrelevant. In reduced game trees, however, where heuristic evaluations are used to approximate game-theoretic values, it might be wise to take into account the amount of good moves available.

The approach by Slagle and Dixon (1970) only differs from the Minimax method in the way how the values of the nodes are computed: MIN is still selecting the move with the lowest value at the min nodes and MAX analogously selects the move with the largest value at the max nodes.

The value of a node is defined either as the value of the best move plus a bonus at max nodes or as the value of the worst move minus a malus[1] at min nodes. The size of the bonus depends on the values of the M best moves in case of a max node. Analogously the size of the malus depends on the N worst moves in case of a min node. (Hence the name M & N search): if the M moves have similar values, the bonus should be large; if the best move differs much from the rest, the bonus should be small (in min nodes analogous reasonings take place to N and the malus notions); if there is only one move at a node, the bonus must be zero; if M = N = 1, the bonus and malus are both zero and the method is equivalent to Minimax. In their description, Slagle and Dixon (1970) mostly dealt with the case for which M = N = 2. In that case, the bonus and malus were taken as a linear function of the difference between the first and second move. They also described a bonus/malus function for M (and N) larger than 2. Their bonus/malus functions, however, seem ad-hoc. M & N search was tested on the game of kalah by Slagle and Dixon, but

[1]Following Uiterwijk and Van den Herik (1994) we call an additional amount that must be subtracted a 'malus' (instead of a bonus as the original authors did).

the search was only 2 ply deep. Therefore, their results are not easy to generalize. M & N search was able to win 0.21 stones on average from Minimax. Applying Minimax with a one-ply deeper search than the opponent gave an advantage of 1.61 stones on average.

Although the notion of an opponent model was not mentioned explicitly by Slagle and Dixon (1970), using a different value for M than for N can be interpreted as such.

### 2.1.3 *-Minimax search for fallible opponents

The opponent model by Reibman and Ballard (1983) assumes that MIN uses the same evaluation function and search method as MAX except that MIN sometimes is fallible: there is a chance at every min node that MIN selects another move than Minimax prescribes. In their model, the probability that MIN selects a specific move depends on the rank of that move (with respect to its value) at the current min node and on the degree of fallibility of MIN.

Let $h$ be a min node and $m(h) = \{m_1, .., m_n\}$ be the moves available at that node. The mapping $r : m(h) \rightarrow \{1, .., n\}$ then gives a rank of the moves according to the value of their subgame: $r(m_i) < r(m_j) \Rightarrow v(h + m_i) \leq v(h + m_j)$. The probabilities that Reibman and Ballard (1983) assigned to the moves is given by the following formula:

$$\Pr(m_i) = \frac{(1 - P_s)^{r(m_i)-1} \cdot P_s}{\sum_j (1 - P_s)^{r(m_j)-1} \cdot P_s} \tag{2.1}$$

Symbol $P_s \in (0, 1]$ represents the fallibility of MIN. If $P_s = 1$, the opponent is not fallible because in this case $\Pr(m_i) = 1$ for the first-rank move. Namely, if $P_s = 1$, the factor $(1 - P_s)$ is zero, but the exponent $r(m_i) - 1$ is also zero for the first ranked move. This means that $(1 - P_s)^{r(m_i)-1} = 1$ for that move. If $P_s$ is close to zero, then all probabilities become almost equal and MIN plays randomly. (The value $P_s = 0$ is not defined, but the limit of $\Pr(m_i)$ for $P_s$ descending to 0 is $1/n$.)

Because MIN is not minimizing anymore but is playing instead by chance, the min nodes of the game tree are replaced by chance nodes. Ballard (1983) introduced a search procedure for game trees that include chance nodes, called *-Minimax search. This search procedure was applied in a simplified form by Reibman and Ballard (1983) to the above situation. Without going into the details of *-Minimax search, we can say that the *-Minimax search method takes the expected value over the moves as the value for chance nodes. This means that the value $v_{rb}(h)$ of a min node is computed as:

$$v_{rb}(h) = \sum_{m_j \in m(h)} \Pr(m_j) \cdot v_{rb}(h + m_j) \qquad \text{(min nodes)} \tag{2.2}$$

(Subscript 'rb' stands for 'Reibman and Ballard'.) For max nodes, *-Minimax search behaves like Minimax search: MAX selects the move that leads to the maximum value and the value of the node becomes:

$$v_{rb}(h) = \max_{m_j \in m(h)} v_{rb}(h+m_j) \qquad \text{(max nodes)} \tag{2.3}$$

Reibman and Ballard (1983) mentioned that their method can be viewed as a special case of M & N search, where M = 1, and N is taken as the number of moves in the position, and where the bonus/malus functions are imposed by the expected values.

There are two problems with the approach by Reibman and Ballard (1983). First, the probability of a node is only assigned on the basis of the *rank* of the move in a particular node and on a global fallibility measure, whereas, for instance, the difficulty of a position is not taken into account. Second, because the expected values in min nodes are used to back-up, that is to compute the values of nodes higher in the tree, the rank of the moves in min nodes is also determined by mistakes that MIN will make deeper in the tree. This would mean that the opponent model of Reibman and Ballard takes into account that MIN is aware of his own errors. In this way, MIN is not only assumed to be fallible, but also to be irrational. Namely, MIN is assumed to know exactly what error he is going to make in the future, but this knowledge includes the ranking of the moves. A rational player would always select the first-ranked move if this knowledge is available. As far as we know, the approach by Reibman and Ballard has never been used in a practical game setting.

### 2.1.4 Jansen's approaches

Jansen (1992a; 1993) described two approaches of using an opponent model: 'Heuristic Program' (HP) and 'Best Optimal' (BO). They were both developed for the special case in which MAX has perfect knowledge of the game but is on the losing side, and MIN can win but is fallible. Such a case could happen in an endgame where only MAX has access to the endgame database. One example is the difficult KQKR endgame in chess (King and Queen versus King and Rook) that has been studied for this special case by Thompson (1986) and Jansen (1992b; 1992c; 1993; and 1992a, chapters 3 and 5). In all these studies, MAX was the computer playing the rook side having access to the endgame database, and MIN was a human who played the queen side, which was obviously the stronger side. We will present some experiments with the same chess endgame in chapter 4.

In practical game-playing, the game-theoretic value of a subgame (e.g., win, draw, or loss) is not the only motivation for selecting a move. For instance, we may distinguish between (1) winning a game whatever the number of moves and (2) winning a game as fast as possible. So, if two moves lead to a win, a player adhering strategy 2 is eager to select the move that leads to the shortest win. Similarly, we may distinguish between (3) accepting a loss immediately and (4) postponing a loss as long as possible. Another example is the 50-moves rule of chess dictating that the game is a draw if during 50 moves no capture has occurred or no Pawn has been moved So, a win has to be secured before the 50-moves rule is applied. Formally, there is no problem with all these four points. Instead of assigning 1 (win), $-1$ (loss) or 0 (draw) to terminal positions of a game, as is normally done, one can assign a payoff value that incorporates the *length* of the history leading to that position. For instance, assign $1000 - |h|$ to a won terminal position and $-1000 + |h|$ to a lost terminal position. Positions in which the 50-moves rule is applied, get assigned the value of 0 (draw). Chess endgame databases usually store the distance from a position to a win (or to the first capture leaving a won position). The distance to win or conversion is also called *the depth* of a position. This information can be translated into a subgame value if careful precautions are taken.

### Jansen's first approach: Heuristic Program

Jansen (1992a; 1993) called his first approach that takes the opponent into account 'Heuristic Program'. Later on, Junghanns (1998) called it Probimax search. It is applied in three variants to the KQKR endgame.

Jansen (1992a; 1993) modelled the opponent by a behaviour strategy (Fudenberg and Tirole, 1991). Such a strategy assigns a fixed probability $\mathrm{Pr}(m_j)$ to every move $m_j$ after a given history $h$. The probability of the moves in Jansen's first approach is determined by a set of chess heuristics but also by the depth of the resulting position. Unfortunately, Jansen did not provide the exact formulae for the probabilities. In passing we note that the probabilities that Reibman and Ballard (1983) assigned to the moves do not constitute a behaviour strategy, because their probabilities depend on the subgames and cannot be assigned just on the basis of the history itself. The formula for the value $v_{hp}(h)$ of a min node $h$ is the same as in formula 2.2:

$$v_{hp}(h) = \sum_{m_j \in m(h)} \mathrm{Pr}(m_j) \cdot v_{hp}(h + m_j) \qquad \text{(min nodes)} \tag{2.4}$$

(Subscript 'hp' stand for 'Heuristic Program'.) MAX is allowed to play suboptimal only to a level of $\delta$, which means that MAX may select a move $m$ of which the subgame value $v(h + m)$ is at most $\delta$ less than the true subgame value $v(h)$. In the approach of Reibman and Ballard (1983), no limit is set to the degree to which the moves could be suboptimal, but Jansen's Heuristic Program (1992a; 1993) puts a bound to the degree of speculation. The value for a max node (cf. formula 2.3) thus is:

$$v_{hp}(h) = \max_{m_i \in m(h) | v(h+m_i) \geq v(h) - \delta} v_{hp}(h + m_i) \qquad \text{(max nodes)} \tag{2.5}$$

### Jansen's second approach: Best Optimal

Jansen (1992a; 1993) assumed that an opponent is more likely to make a mistake when the ratio between the optimal moves and available moves is lower. MAX uses this knowledge to choose between optimal moves. The second approach is therefore called: 'Best Optimal'. Let $opt(h) \subseteq m(h)$ be defined as:

$$opt(h) = \{m_j \in m(h) \mid v(h + m_j) = v(h)\} \tag{2.6}$$

then MAX will select the following move:[2]

$$m = \min_{\substack{m_i \in m(h) \mid \\ v(h+m_i)=v(h)}} \arg \frac{\#opt(h+m_i)}{\#m(h+m_i)} \tag{2.7}$$

Junghanns (1998) gave the name 'Speculative Play' to the second approach of Jansen (1992a; 1993). Strictly spoken, there is no speculation in this approach: MAX is still playing optimally, in contrast to Jansen's first approach, in which MAX is allowed to speculate. We will use this name for the next approach instead.

### 2.1.5 Speculative Play

Uiterwijk and Van den Herik (1994) designed the Speculative Play approach for the case in which MAX knows the true subgame values for at least some part of the game tree. This will usually happen towards the end of a game. The true subgame values in this approach are assumed to include the distance to win, as in Jansen's (1992a; 1993) approaches.

Uiterwijk and Van den Herik (1994) aimed, like Jansen (1992a; 1993), to exploit fallibility of the opponent. Assume that an opponent is presented two positions that theoretically lead to a draw. In the first position, only one move can lead to a draw and the others lead to a sure loss for the opponent. In the second position, almost all moves can lead to a draw, and only few moves lead to a loss. A perfect opponent will select the correct move in both cases but a fallible opponent has a higher probability to lose in the first position. MAX should therefore prefer the first position rather than the second position. This motivation is fundamentally the same as in the approach by Reibman and Ballard (1983) and in the Best-Optimal approach of Jansen (1992a; 1993), but the domain of application and the formulae are different. Uiterwijk and Van den Herik introduced a *bonus/malus* function to express the preferences between equal-valued moves. Uiterwijk and Van den Herik implicitly assumed that MIN is also speculating about errors of MAX, which was the case in M & N search too. Speculative Play is given by:

$$v_{sp}(h) = \begin{cases} \max\limits_{m_j \in m(h)} v_{sp}(h+m_j) + bonus(h+m_j) & \text{(max nodes)} \\ \min\limits_{m_j \in m(h)} v_{sp}(h+m_j) - malus(h+m_j) & \text{(min nodes)} \\ V(h) & \text{(leaf nodes)} \end{cases} \tag{2.8}$$

$$bonus(h) = \min(w_{sp} \cdot \frac{\sum\limits_{\substack{m_j \in m(h) \mid \\ d(h+m_j)<\infty}} d(h+m_j)}{\#\{m_j \in m(h) \mid d(h+m_j) = \infty\}}\ , \ limit_{sp}) \tag{2.9}$$

The function *malus* is defined analogously. The function $d(h)$ returns the distance to win from $h$, or $\infty$ if that distance is not known yet. The bonus function is

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

only defined for nodes in which there is at least one move that does not lead to a certain win. The factor $w_{sp}$ is used to scale the function to the range of $v(h)$; the factor $limit_{sp}$ is the maximum level of speculation. Observe that $limit_{sp}$ plays the same role as the factor $\delta$ in Jansen's (1992a; 1993) Heuristic Program: it indicates by how much the value of $v_{sp}(h)$ is allowed to deviate from $v(h)$. However, the effect is different because in the Heuristic-Program approach, the true value of $h$ is known, whereas in this approach, only the value of a reduced subgame at $h$ is known. The factors $limit_{sp}$ and $w_{sp}$ can also be used to model the level of fallibility of the opponent: the weaker the opponent, the higher these factors will be. This behaviour is comparable to the factor $P_s$ in the approach of Reibman and Ballard (1983).

Next to opponent modelling, the inventors of Speculative Play also aimed to improve search on another point. To distinguish between true payoff values and heuristic values, the true values are always larger (in absolute value) than the heuristic values (see also section 4.1.1). One important exception is the value for draw (0) which lies within the range of heuristic values. This means that a subgame value of zero in a reduced game does not have to mean that the true value of a position is also a draw. When a zero appears isolated among siblings, there is a large possibility that the true value of that node is not zero. This can be exploited by Speculative Play. Uiterwijk and Van den Herik applied Speculative Play on a test set in chess and indeed observed a small increase in the number of positions that could be solved. Currently, research is going on by Kajihara, Hashimoto, and Iida (2002) on Speculative Play in the endgame of shogi.

### 2.1.6 Comparison of the approaches

The five approaches were developed for different stages of a game, which makes it difficult to compare them. Slagle and Dixon (1970) and Reibman and Ballard (1983) did not restrict their approaches to a specific part of the game, although they did not seem to take the endgame explicitly into consideration. Jansen (1992a; 1993) concentrated on the endgame in which only MAX has access to an endgame database. Uiterwijk and Van den Herik (1994) studied the phase directly preceding the endgame: for some parts of the tree the true game value is known.

All five approaches take a fallible opponent into account, thereby using some model of the opponent. Below we answer the question how the opponent model is implemented in each of the approaches:

| | | |
|---|---|---|
| M & N Search | : | bonus/malus depending on a subset of moves. |
| *-Minimax Search | : | move probabilities based on move result. |
| Heuristic Program | : | move probabilities based on heuristics and move result. |
| Best Optimal | : | bonus depending on the relative number of optimal moves. |
| Speculative Play | : | bonus/malus depending on the relative number of certain wins. |

So, either bonuses/maluses or move probabilities are used to model an opponent. However, not all opponents are equal. Therefore the next question to be answered is: how should the opponent model be parametrized for a specific opponent in each of the five approaches above.

| | |
|---|---|
| M & N Search : | by the difference between M and N. |
| *-Minimax Search : | by the degree of fallibility, $P_s$. |
| Heuristic Program : | by the selection of move heuristics used. |
| Best Optimal : | not; all opponents are assumed to behave equally. |
| Speculative Play : | by the degree of fallibility in $w_{sp}$ and $limit_{sp}$. |

Next to an opponent model, the five approaches each have their own level of speculation. The last question therefore is: what level of speculation is allowed, that is, how much is MAX allowed to deviate from normal play.

| | |
|---|---|
| M & N Search : | unlimited. |
| *-Minimax Search : | unlimited. |
| Heuristic Program : | at most $\delta$ below the true value. |
| Best Optimal : | no speculation. |
| Speculative Play : | at most $limit_{sp}$ above (below) value for bonus (malus). |

The speculation in Jansen's Heuristic-Program (1992a; 1993) approach differs from that in Uiterwijk and Van den Herik's (1994), because in the Heuristic Program, MAX knows exactly how much risk is being taken in terms of the increased distance to win. In Speculative Play, MAX only knows how small the difference between the reduced-game values of the best and second-best move has to be before a bonus can cause the second-best move to be selected. There is no way in which MAX can measure the real risk at stake.

None of these five approaches have yet been applied in real competition although some of the current top-range computer-chess programs claim to use a form of speculative search. The authors of HIARCS (Mark Uniacke[3]) and GAMBIT TIGER 2.0 (Christophe Théron[4]) claim that their programs perform true speculative sacrifices. There is no information published on how this is implemented and performed, nor is there information published on how the opponent is modelled.

### 2.1.7 Use of opponent knowledge in chess tournaments

During today's computer-chess tournaments such as the World Computer Chess Championship of the ICGA[5], the authors of chess programs often adapt their program to a specific opponent. There are three elements in a computer-chess program that usually can be tuned to a specific opponent before a game starts: the opening book, the evaluation function, and the contempt factor. An opening book contains the moves at the start of the game that are to be played by the program. When the characteristics of a given opponent are known, for instance through previously played games, it is possible to build an opening book in a such way that the opponent is lured into a position that has proven to be more difficult for that opponent. There are programs commercially available that can generate such opening books (e.g. CHESS ASSISTANT 6.1[6]).

[3]See: http://www.chess4less.com/2-Hiarcs.htm.
[4]See: http://www.chessbase.com.
[5]*International Computer Games Association*, see www.icga.org.
[6]See: http://icc.convekta.com/CAFamily/CA6.

The second element of a computer-chess program that can be tuned to an opponent is the evaluation function. Many chess programs (e.g., CRAFTY, CHESSMASTER, CHESS TIGER) offer the possibility to adjust some of the high-level parameters of the evaluation function. Changing these parameters influences the playing style of the program, for instance, to play more aggressively or more defensively. When the playing style of an opponent is known, one can select these parameters carefully in answer to the specific style.

The third element is the contempt factor. This is one of the first opponent-modelling mechanisms in computer chess. The contempt factor deals with an evaluation value at which a draw is accepted. Usually, this value is set at zero. However, it might happen that a threefold repetition occurs due to a zero evaluation which is the highest evaluation among all variants. Against a strong opponent it might be worthwhile to accept a draw even at a technically won position (thus even with an evaluation value $> 0$) and against a weak opponent, a draw would preferably be avoided even when a position seems to be lost (thus even with an evaluation value $< 0$). The contempt factor is often regarded as a synonym for *draw score* and is as such implemented as part of the evaluation function. Most current chess programs offer the user an option to set a contempt factor at the beginning of a game.

In all three cases above, the opponent model is not explicit. The program does not know how the opponent will behave, it is only adjusted in such a way that the author expects the program to perform better.

## 2.2 The Invention of Opponent-Model Search

The example of nim-5 in chapter 1 clearly shows that to have explicit knowledge of the opponent's strategy can be beneficial: a lost game is turned into a won game. In general, when the opponent's (MIN) strategy is fully known to a player (MAX) in a perfect-information zero-sum game and the strategy of MIN is not optimal, then MAX can always obtain a payoff that is at least as high, but often higher than when this knowledge is not available. This is true because of the definition of the Minimax solution.

Such a situation is highly hypothetical in two respects: (1) MAX must know MIN's strategy precisely and (2) MAX must be omniscient in the game because he must foreknow all the effects of a move. The first condition is difficult to achieve in practice and the second condition can in principle only be achieved in the final stage of a game when the player has access to a suitable endgame database.

In the introduction to his thesis, Jansen (1992a) analyzed under which conditions explicit knowledge of the opponent's strategy can be exploited. Jansen assumed that MAX is omniscient and knows the exact game-theoretic values of all positions. Player MIN is fallible and makes errors with a given probability $p$. Player MAX is not totally aware of MIN's strategy and makes a prediction error $\epsilon$. In this setting, Jansen showed that speculating on the opponent's move is only useful when the prediction error $\epsilon$ is not too high and when MIN does not make too few errors ($p > 0$) but also not *too many* ($p < 1$). If the prediction error $\epsilon$ is zero, the formulae of Jansen correctly indicate that speculation on the opponent model is always useful.

As already discussed in chapter 1, computer game-playing consists of finding solutions to a series of reduced perfect-information games in which the true payoffs are (partly) replaced by heuristic values. Normally, game-tree search methods assume that both players behave rationally and therefore apply a Minimax strategy to this reduced game. Rationality means that a player will always select the move of which he believes that it will lead to the highest payoff. However, to assume that players behave rationally is not sufficient to justify the Minimax strategy. Both players must know that the other player behaves rationally. To put it more precisely, it must be *common knowledge* that both players behave rationally.

### 2.2.1 Haifa, Israel

Carmel and Markovitch (1993; 1996a) developed at Technion, in Haifa – Israel, a generic search method called M *search* in which MAX is assumed to have perfect knowledge of MIN's strategy and payoffs, but MIN does not necessarily know MAX's. Both players are assumed to play rationally, and both player assume that this is common knowledge.

The generic M-search approach is parametrized by MIN's strategy. When MIN is using the same evaluation function as MAX and is assuming that MAX does likewise, then M search is called M$^0$ *search* and the method is equal to Minimax search.

The next logical step was to assume that MIN is using an other evaluation function ($V_{op}$) than MAX and that MIN assumes that MAX is using this other evaluation function $V_{op}$. In other words, MIN is assumed to use a Minimax strategy on a different version of the reduced game. Carmel and Markovitch call this version M$^1$ *search*.

M$^1$ search was developed as an intermediate step towards an approach in which the opponent model is expanded in a recursive epistemic manner. This approach, called M$^*$ *search*, will be treated in section 2.4.4 below.

### 2.2.2 Maastricht, The Netherlands

At the same period that Carmel and Markovitch developed M search in Israel, a visitor from Japan, Iida, worked in the Netherlands at Universiteit Maastricht together with Uiterwijk, Van den Herik, and Herschberg on an approach that is basically the same as M$^1$ search (Iida, Uiterwijk, and Van den Herik, 1993a). The authors call their approach *Opponent Model (OM) search*, a name that we also use in this thesis.

The developers of OM search stated as a precondition for their approach that MAX is completely aware of MIN's strategy and that this strategy is fully determined by the evaluation function of MIN. Implicitly, the authors assumed that MIN uses a Minimax-equivalent search method.

## 2.3 OM-Search Basics

In this section we will first give the original description of the OM-search method and a small example. We will then produce an enhanced version of this description as intended by the authors. Next, we will briefly deal with some basic properties of the method as presented in Carmel and Markovitch (1993) and Iida *et al.* (1993a). We discuss the issue of risk and possible gain in OM search and finish the section with some proposed application areas.

### 2.3.1 Original formulation

OM search is described originally by the equations below (in the notation of chapter 1). In these equations, $V_0(\cdot)$ is the evaluation function that MAX is using and $V_{op}(\cdot)$ is the evaluation function for MIN. The node values computed for the MAX player are indicated by $v_0(\cdot)$ and for the MIN player by $v_{op}(\cdot)$.

$$v_0(h) = \begin{cases} \max\limits_{m_j \in m(h)} \; v_0(h+m_j) & p(h) = \text{MAX} \\ v_0(h+m_j), \quad m_j = \min\limits_{m_i \in m(h)} \arg \; v_{op}(h+m_i) & p(h) = \text{MIN} \\ V_0(h) & h \in E \end{cases} \tag{2.10}$$

$$v_{op}(h) = \begin{cases} \max\limits_{m_j \in m(h)} \; v_{op}(h+m_j) & p(h) = \text{MAX} \\ \min\limits_{m_j \in m(h)} \; v_{op}(h+m_j) & p(h) = \text{MIN} \\ V_{op}(h) & h \in E \end{cases} \tag{2.11}$$

Observe that the value of $v_{op}(h)$ is exactly the Minimax value (or subgame value) of $h$ for MIN.

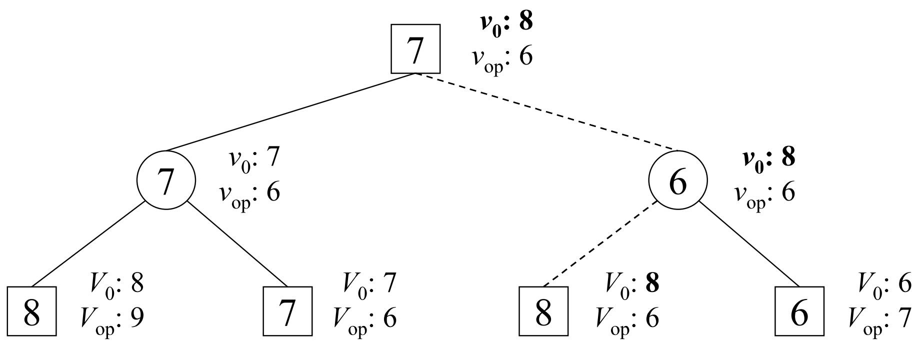


Figure 2.1: An OM-search example. The numbers inside the nodes indicate the Minimax values. Next to the nodes are the values for $v_0$ ($V_0$) and $v_{op}$ ($V_{op}$).

Figure 2.1 gives a small example of how OM search works. Notice that at leaf nodes (which are max nodes), the value of $v_0(\cdot)$ is equal to $V_0(\cdot)$, which is also the Minimax value for the leaf nodes. At the left min node, the OM-search value ($v_0$) is equal to the Minimax value for MAX (7) because MIN selects the same branch (right) as MAX would do. From the right min node, however, MAX would be led via the right branch if Minimax was used, but because MIN's evaluation for the left branch ($V_{op} = 6$) is lower than his evaluation for the right branch ($V_{op} = 7$), MAX may assume that MIN chooses the left branch (dotted) and so MAX receives a higher value ($v_0 = 8$, in bold) than the Minimax value (7). Hence, at the root, MAX chooses the right branch (dotted) instead of the left one, which is the Minimax principal variation. Of course, there is a risk that MAX has a wrong impression of the opponent. If the opponent is actually using $V_0$ instead of $V_{op}$, then MAX's value of the tree is only 6 instead of 8 (or instead of 7).

Although the terminology and notation for M$^1$ search is rather different from that of OM search, both approaches are the same. In both cases, MAX is assumed to behave rationally in the context of the reduced game, trying to maximize the own payoff and being aware of the complete strategy of MIN. At the other side, MIN is also assumed to play the same truncated game rationally but with another evaluation function and unaware of MAX strategy, assuming that MAX uses MIN's evaluation function.

### 2.3.2 Enhanced formulation

The formulations for OM search in the previous section do not cover the intentions of the developers completely because there are two issues not included. The first issue occurs when MIN evaluates two (or more) moves equally (i.e., $v_{op}(m_i) = v_{op}(m_j)$, $i \neq j$, $m_i, m_j \in m(h)$). In such case MAX has to select either $m_i$ or $m_j$. Most safely, MAX should select the move with the lowest own evaluation.

The second issue is that the formulation in the previous subsection suggests that MAX and MIN use exactly the same set of leaf nodes. This needs not be the case, because MIN might have more limited resources than MAX or might use different rules to stop searching at a given node. The developers of OM search implicitly assume that MAX will use Minimax search on subgames that are out of reach for MIN. It is not difficult to incorporate this into the formulae for OM search. Let $H_0$ denote the set of histories that MAX is using and $H_{op}$ the set for MIN. The leaf nodes for both players are $E_0$ and $E_{op}$. All together, the issues lead to the following enhanced formulae for OM search:

$$v_0(h) = \begin{cases} \max\limits_{m_j \in m(h)} v_0(h + m_j) & p(h) = \text{MAX} \\ \min\ v_0(\min \mathop{\arg}\limits_{m_i \in m(h)} v_{op}(h + m_i)) & p(h) = \text{MIN} \wedge h \in H_{op} \\ \min\limits_{m_j \in m(h)} v_0(h + m_j) & p(h) = \text{MIN} \wedge h \notin H_{op} \\ V_0(h) & h \in E_0 \end{cases} \tag{2.12}$$

$$v_{op}(h) = \begin{cases} \max\limits_{m_j \in m(h)} v_{op}(h + m_j) & p(h) = \text{MAX} \\ \min\limits_{m_j \in m(h)} v_{op}(h + m_j) & p(h) = \text{MIN} \\ V_{op}(h) & h \in E_{op} \end{cases} \tag{2.13}$$

### 2.3.3 Two derived characteristics

Two theoretical characteristics of OM search are described by both Carmel and Markovitch (1993) and Iida *et al.* (1993a). The first characteristic of OM search that both groups describe is that the root value of a tree traversed by OM search is equal or higher than the (Minimax) game value of that tree. In chapter 5 we will show that this property holds for a large class of game-tree search algorithms.

The second characteristic is that the value of the root only differs from the Minimax value if somewhere on the principal variation of the tree, MIN selects a move that is suboptimal according to MAX's evaluation function. Iida *et al.* called these deviations *errors of the opponent*, but unless MAX is omniscient, the term 'error' is not appropriate. We will give an alternative description of these deviations in chapter 3.

By providing these characteristics, the inventors of OM search suggest that OM search allows for exploiting errors of the opponent. Neither characteristic, however, provides us with any clue on how the quality of play improves when OM search is used, unless MAX is omniscient.

### 2.3.4 Risk and possible gain

An obvious risk that occurs when OM search is applied is caused by errors that MAX makes in the prediction of MIN's strategy. Although Carmel and Markovitch (1993; 1996a) also dealt with prediction errors (or *modelling errors* as they named it) by MAX, the analysis by Iida *et al.* (1993a; 1994) of prediction errors is more elaborate.

*Risk* is defined by Iida *et al.* as the difference between the obtained payoff for MAX when using Minimax and the payoff for MAX when sticking to the OM search strategy while MIN is in fact using a Minimax strategy with MAX's evaluation function. In other words, the risk is the difference between the payoff that MAX can guarantee (using Minimax) and the worst that can happen if MAX sticks to the prediction of MIN while MIN is behaving differently. Let $s_{mm(V)}$ denote a Minimax strategy, using evaluation function $V$ and $s_{om(V_1,V_2)}$ an OM-search strategy with evaluation functions $V_1$ and $V_2$. In the notation of chapter one, the risk using OM search can be expressed as:

$$Risk = v(\langle s_{mm(V_0)}, s_{mm(V_0)} \rangle) - v(\langle s_{om(V_0,V_{op})}, s_{mm(V_0)} \rangle) \tag{2.14}$$

This risk can be computed not only for the root node, but for every max node in the game tree.

Iida *et al.* (1993a; 1994) also discussed the *possible gain* of OM search. This is a measure of how much MAX could benefit from using OM search. It is defined as the difference between the OM-search value of the game tree and the Minimax value:

$$PossibleGain = v(\langle s_{om(V_0,V_{op})}, s_{mm(V_{op})} \rangle) - v(\langle s_{mm(V_0)}, s_{mm(V_0)} \rangle) \tag{2.15}$$

The same definition is used to express the possible gain at every max node in the game tree.

Both risk and possible gain are properties of the reduced game and not of the actual full game. This means that the usefulness of these measures is practically limited to cases in which MAX is omniscient. In section 4.2 we will show that there exist another type of risk that is caused by errors in MAX's evaluation function.

### 2.3.5 Application areas

As we already mentioned, Jansen (1992a) indicated under which conditions speculating on the opponent's behaviour can be beneficial. Carmel and Markovitch (1993; 1996a) interpreted these conditions and formulated three types of positions in which OM search could be of use. The first type is a *swindle* position in which MAX expects MIN to underestimate a good move and will play a poor move instead. The second type is a *trap* position in which MAX expects MIN to overestimate a move. The third type of position is a loosing position where there are more than one moves that lead to a loss. Instead of selecting an arbitrary move (as Minimax would do), OM search can use the opponent model to select a swindle move.

Carmel and Markovitch (1993; 1996a) suggested that there is also another situation in which opponent models might be of use. At the introduction of reduced games in chapter 1, we assumed that the heuristic evaluation function is symmetric, that means that the value for the two players is opposite for all positions. It can, however, happen that this is not the case because the heuristic evaluation function contains factors that are beneficial to *both* players at the same time. In such case, Minimax will not predict the moves of the opponent correctly. Carmel and Markovitch claimed that their approach can be applied in this case. In section 2.5 we will return to this matter.

Iida *et al.* (1995; 1997b) gave some other areas of application. The first is in a tutoring environment (Iida, Handa, and Uiterwijk, 1995). By using an opponent model of the pupil, a program could develop a tutoring strategy that leads the pupil to find a good move 'by accident'. Another area of application that the authors provided is in the field of coalition games like bridge where opponent models can be used to model the partner. In this way, a co-operative strategy could be developed (Iida, Uiterwijk, and Van den Herik, 1997b).

## 2.4 Developments in OM-Search

The description of OM search as given above is denoted by Carmel and Markovitch as $M^1$ search and by Iida *et al.* as *pure* OM search. Both groups have produced variants of the basic search method. In this chapter we will enumerate and discuss all known variants. In chapter 5 we will introduce yet another variant of OM search that is called *probabilistic* OM search.

### 2.4.1 Pure OM search

Iida *et al.* (1993b) gave two implementations of pure OM search. The first algorithm is called $\beta$-pruning OM search. This algorithm computes $v_0$ and $v_{op}$ in one pass and is able to prune the tree at max nodes. Pruning at min nodes is not possible. In chapter 3 we will return to this algorithm. The second algorithm is called Root-value Pruning OM search. This is a two-pass algorithm. In the first pass, the Minimax value of the root is determined (based on $V_0$). In the second pass, the OM-search value is determined, but the search tree is pruned at max nodes as soon as at least the root value can be obtained. This algorithm appears to be more efficient than $\beta$-pruning, but unfortunately, the root-value pruning does not necessarily produce the same outcome as OM search because it could fail to find all possibilities to utilize MIN's weaknesses.

## 2.4.2 OM search with risk management

Above we gave the definitions for risk and possible gain in OM search. Iida *et al.* (1993a; 1997a) defined three variants of OM search that explicitly deal with risk and possible gain.

The first variant is called OMGF search (OM Gain First). In this variant, MAX first selects moves at a max node that maximize the possible gain of the max node. If there is more than one move with the same maximal possible gain, MAX selects a move that minimizes the risk.

The second variant is called OMLF search (OM Loss First). This variant is the opposite: MAX first selects moves at a max node that minimize the risk of the max node. If there is more than one move with the same minimal risk, MAX selects a move that maximizes the possible gain.

The third variant, OMB search (OM Balanced), combines OMGF search and OMLF search. In this variant, MAX selects the move at every max node that maximizes the following factor ($\delta$ is used to balance between possible gain and risk):

$$\epsilon(h) = \frac{PossibleGain(h)}{Risk(h)\ +\ \delta} \tag{2.16}$$

The authors did not indicate how these methods can be implemented in an efficient way.

## 2.4.3 $(D,d)$-OM search

Iida *et al.* (1993b; 1994), and Gao *et al.* (1999; 2001) described a variant of OM search called $(D,d)$-OM search. In this variant, the focus is on the difference in search depth between MAX and MIN: while MAX is searching with fixed depth $D$, MIN is assumed to search only with fixed depth $d$, where $d \leq D-2$. For larger values of $d$, applying this method is not useful, according to Iida *et al.* (1993b). Strictly spoken, $(D,d)$-OM search is not a variant of OM search, but a specialization of this method because the formulae 2.12 and 2.13 cover $(D,d)$-OM search completely. The definition of $(D,d)$-OM search in Gao *et al.* (2001) includes the possibility for MIN and MAX to have different evaluation functions, but in all examples and experiments both players used the same evaluation function.

The authors gave an implementation of $(D,d)$-OM search that is called $\alpha$-$\beta^2$ pruning OM search. This algorithm is a one-pass algorithm, like $\beta$-pruning. The main difference lies in the lower parts of the search tree in which MAX is still searching, but MIN has reached the search horizon. In this part of the tree, $\alpha$-$\beta$ search is used to compute the value of the nodes for MAX. We will discuss this algorithm in more detail in section 3.1.3.

Gao *et al.* (2001) performed some experiments with $(D, d)$-OM search on random game trees, and also in the game of othello. In the latter experiments MAX had perfect knowledge of MIN's evaluation function, so MAX was able to predict the moves of MIN without any error. The performance of $(D, d)$-OM search was compared with plain Minimax search with search depth $D$. The results showed that $(D, d)$-OM search seemed to outperform Minimax search (but no proof of statistical significance is provided). With larger difference between $D$ and $d$, the distance between the performance of $(D, d)$-OM search and Minimax search increased to a maximum of 4 per cent.

### 2.4.4 M* search

As already mentioned, Carmel and Markovitch (1993; 1994; 1996a; 1996b; 1998) provided a search method that generalizes OM search, called M* search. In this method, MAX has an opponent model of MIN that possibly includes an opponent model of MAX, which can include a second-order opponent model of MIN, and so on. The assumed knowledge of the players is asymmetric: MAX has full knowledge of MIN's strategy, including the opponent model that MIN is using, but this is not the case for MIN. Furthermore, the opponent models are used at different time points in the game: at the root node, MAX is predicting the moves of MIN at the min nodes just below the root, knowing that MIN is predicting MAX's moves at the max nodes just below that, and in turn knowing that MAX is predicting MIN's moves on the next layer of min nodes, and so on. So the maximum number of nested opponent models that is used is equal to the search depth. Most of the times, the number of nested opponent models will be restricted to a few. The inner most opponent model will normally be a Minimax strategy.

The opponent models in M* search are determined by evaluation functions $V_0$, $V_1, \ldots, V_n$, where $V_0$ is MAX's evaluation function, $V_1$ the evaluation that MIN is using, $V_2$ the evaluation function that MIN thinks MAX is using, until the inner-most opponent model $n$ is reached, which is assumed to be a Minimax strategy. M* search then is described by the following formulae:

$$\underset{0 \le i < n}{v_i(h)} = \begin{cases} \max\limits_{m_j \in m(h)} \; v_i(h + m_j) & p(h) = p(V_i) \\ v_i(h + m_j), \quad m_j = \max\limits_{m_k \in m(h)} \arg \; v_{i+1}(h + m_k) & p(h) \neq p(V_i) \\ V_i(h) & h \in E \end{cases} \tag{2.17}$$

$$v_n(h) = \begin{cases} \max\limits_{m_j \in m(h)} \; v_n(h + m_j) & p(h) = p(V_n) \\ \min\limits_{m_j \in m(h)} \; v_n(h + m_j) & p(h) \neq p(V_n) \\ V_n(h) & h \in E \end{cases} \tag{2.18}$$

We used the notation $p(V_i)$ to indicate the player that is assumed to be using evaluation function $V_i$. So, $p(V_0)$ is MAX, $p(V_1)$ is MIN, *et cetera.* When $n = 1$, the formulae coincide with those of OM search (2.10, 2.11) and when $n = 0$, the formulae describe Minimax search because formula (2.17) is not applicable in that case.

Carmel and Markovitch (1996a) provided a number of algorithms for this search method. The first algorithm (also called M$^*$ search) is a multi-pass version in which at every max node, the algorithm is called recursively, once with the opponent model only, and once with the own evaluation function together with the opponent model. The first call strips away the own evaluation function, unveiling the next level of opponent model. As a result, the number of calls to the algorithm increases exponentially with the number of nested opponent models.

The second algorithm that Carmel and Markovitch (1996a) provided is a one-pass algorithm (called M$^*_{1\text{-}pass}$ search). This algorithm returns a vector of values at every max node, one value for every nested opponent model. The number of node expansions in the one-pass algorithm is smaller than in the multi-pass version, but the number of evaluations is larger.

These two algorithms can be augmented with 'simple pruning' by replacing search at the innermost model level by $\alpha$-$\beta$ search. When M$^*$ search coincides with OM search, the algorithm M$^*_{1\text{-}pass}$ search with simple pruning is equal to the $\beta$-pruning algorithm of Iida *et al.* (1993b).

The third and fourth algorithms incorporate pruning of the search tree. The algorithms are called $\alpha$-$\beta^*$ search and $\alpha$-$\beta^*_{1\text{-}pass}$ search. Pruning in these algorithms is based on bounds on the difference between opponent models. We will go into more detail on these algorithms in subsection 3.2.4.

Carmel and Markovitch (1996a) also gave a (nameless) algorithm that takes differences in search depth into account. They do not provide any means of pruning in this algorithm, but it must be possible to add pruning in an analogous manner as in $\alpha$-$\beta^2$ pruning OM search.

Some experiments with M$^*$ search were conducted in the game of checkers (Carmel and Markovitch, 1996a, 1998). In these experiments, the authors tested four aspects: (1) difference in search depth between MAX and MIN, (2) difference in evaluation function, (3) error in the modelled search depth, and (4) error in the modelled evaluation function. The opponent model used in the experiments consisted of only one level, which makes the experiments comparable to the experiments by Gao *et al.* (2001) in the previous subsection.

When MAX had correct knowledge of MIN's strategy, the results showed that M$^*$ search outperformed Minimax search, similar to $(D, d)$-OM search. When the difference in search depth increased, the effect increased and likewise when the difference in evaluation quality increased. The results of experiments with erroneous opponent models showed that using these models can be harmful. Some of the experiments in chapter 5 we will also show this effect of OM search.

### 2.4.5 $\mathrm{M}^*_\epsilon$ search

In order to cope with modelling errors, Carmel and Markovitch (1996a) developed a variant called $\mathrm{M}^*_\epsilon$ search that allows some bounded error $\epsilon$ in the opponent model. Instead of assuming a given value $v(h)$ for the evaluation of a node $h$, the method allows an error of $\epsilon$ such that the real value of the opponent's evaluation must lie in the interval $[v(h) - \epsilon, v(h) + \epsilon]$. The algorithm is analogous to the $\mathrm{B}^*$ algorithm of Berliner (1977). When $\epsilon$ approaches $\infty$, $\mathrm{M}^*_\epsilon$ search behaves equal to Minimax search. The authors do not provide any pruning details, nor any experimental results for this algorithm.

## 2.5 Opponent Models in Game Theory

In this section we provide a brief overview of topics from modern game theory that are related to the search methods discussed in this chapter. The goal of this section is to point out that in mathematical and economical game theory researchers have dealt with the issue of opponent models too. The solutions that game theory offers differ from the solutions above, but there are common elements. We are not able and do not want to give a complete account on modern game theory; for that we refer, for instance, to the textbook by Fudenberg and Tirole (1991).

### 2.5.1 Nonzero-sum perfect-information games

Zero-sum perfect-information games were already introduced in chapter 1. They have a long history in game theory. Zermelo (1913) discussed games that nowadays would be called finite zero-sum perfect-information games. According to Van den Herik (1983), Zermelo mainly studied the way in which one player can force a win or postpone a loss in a game like chess. He provides an algorithm to solve this types of games, which is basically the Minimax algorithm, also known as the Zermelo-Kuhn or backward-induction algorithm. Von Neumann (1928) studied games in which the payoffs are not simply wins or losses as in Zermelo's paper. Furthermore, von Neumann was interested in the *strategies* that players have to adopt in order to maximize their payoff. He proved that the Minimax algorithm of Zermelo indeed produces this strategy in case of zero-sum perfect-information games. The work by Zermelo and von Neumann constitutes the main body of game theory on which research in computer-game theory is based. It is, however, interesting to travel further in the history of game theory.

The idea of an optimal Minimax strategy was generalized by Nash (1950) for games in which the players do not necessarily have opposite payoffs. He defined the concept of *equilibrium* in which two players have a strategy such that neither one can on his own increase the own payoff. In any perfect-information game, such an equilibrium point must exist and often there will be multiple equilibria possible. Selten (1965) showed that not all of these Nash equilibria are equally likely when the players behave truly rationally. Nash equilibria that are *subgame perfect* are preferred. Subgame perfect means that the strategies of both players must be such that they form Nash equilibria in all (proper) subgames. For zero-sum games, all

Nash equilibria appear to have this property, so the Minimax algorithm is still producing preferable strategies. (Both Nash and Selten received a Nobel price in economic sciences for this work in 1994.) The case is different, however, for nonzero-sum games.

A two-player nonzero-sum game with perfect information can be observed as a game in which both players have a perfect opponent model of each other. The difference with the OM-search approach is that the knowledge is *symmetrical*: both players know each others model, and furthermore, they are aware of the fact that the opponent has such a model. The subgame-perfect equilibria for this types of games can be found by backward induction, which can be described by:

$$v_0(h) = \begin{cases} \max\limits_{m_j \in m(h)} \; v_0(h + m_j) & p(h) = \text{MAX} \\ v_0(h + m_j), \quad m_j = \max\limits_{m_i \in m(h)} \text{arg} \; v_{op}(h + m_i) & p(h) = \text{MIN} \\ V_0(h) & h \in E \end{cases} \tag{2.19}$$

$$v_{op}(h) = \begin{cases} \max\limits_{m_j \in m(h)} \; v_{op}(h + m_j) & p(h) = \text{MIN} \\ v_{op}(h + m_j), \quad m_j = \max\limits_{m_i \in m(h)} \text{arg} \; v_0(h + m_i) & p(h) = \text{MAX} \\ V_{op}(h) & h \in E \end{cases} \tag{2.20}$$

Since $V_0$ and $V_{op}$ are not opposite, the 'min arg' term is replaced by a 'max arg' term. The search method can be implemented in a one-pass algorithm, but pruning might be an obstacle.

Carmel and Markovitch (1996a) used an opponent model in a setting that is in fact better suited for the nonzero-sum approach than for M* search. In their experiments on checkers, both players used the same evaluation function which included a factor that both players wanted to reduce, namely the total number of figures on the board:

$$f(board, player) = Material(board, player) - 0.004 \cdot TotalFigures(board) \tag{2.21}$$

Although both players used the same evaluation function, the Minimax approach assumed that MIN was using the opposite of MAX's evaluation. This gave the following evaluation function for MIN:

$$f(board, player) = Material(board, player) + 0.004 \cdot TotalFigures(board) \tag{2.22}$$

Clearly, when using Minimax, MAX assumed that MIN was trying to *increase* the total figures on the board. In the experiments, MAX was assumed to know the correct evaluation function of MIN, but MIN was playing Minimax. It is more realistic to assume that MAX and MIN treat each other equally, since they both use the same evaluation function and both know that the other does so, too. It seems that this game setting is more adequately modelled by a nonzero-sum game with perfect information.

Using bilateral opponent modelling in the form of a nonzero-sum game eliminates the idea of MAX exploiting weaknesses in MIN's strategy. Both players are trying to get the best heuristic value and know that the other is also trying to do so.

## 2.5.2 Problems with subgame-perfect equilibria

Selten's (1965) motivation for the preference of subgame-perfect equilibria is based on both players behaving rationally. Rational is defined as to select always the move that maximizes the own payoff. Aumann (1995) showed that rationality alone is not sufficient to obtain subgame-perfect equilibria. However, he proved that when the rationality of both players is *common knowledge* at the start of the game, this necessarily will lead to backward induction and to a subgame-perfect equilibrium.

Some scientists (see for instance, Binmore, 1996, and Stalnaker, 1996) had difficulties with accepting this conclusion because of the definition of common knowledge of rationality. One problem is caused by counterfactuals: when we look at node ***c*** in the game tree of figure 2.2, backward induction tells that MIN should select the left branch (***f***), because MAX is assumed to behave rationally at nodes ***f*** and ***g***. However, if node ***c*** were to be reached, MAX would have proven not to behave rationally. At this point, MIN could have changed his opponent model of MAX and believe that MAX would choose moves leading to ***o*** instead of ***n***, which would mean that MIN should select the right branch at node ***c***. At some point, this kind of reasoning can cause MAX to select another move at the root. The way in which common knowledge of rationality is defined and dealt with appears to be important for the equilibrium that is reached. It does not have to be the subgame-perfect one, even in zero-sum games of perfect information.

The counterfactual at node ***c*** needs not to be interpreted as an irrational act of MAX. Another explanation could be that MAX made a simple mistake at that point, but it is still common knowledge that MAX is behaving rationally. Selten (1975)

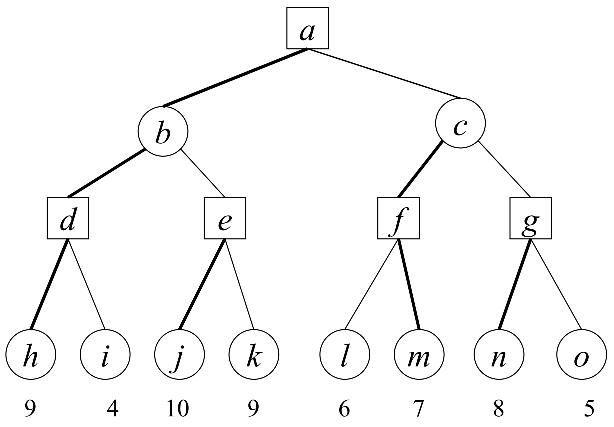


Figure 2.2: A game tree to illustrate the counterfactual problem. The bold lines indicate the Minimax strategy for both players.

extended this idea and stated that players might have a small but positive chance to make mistakes. He called a game which includes such mistakes a perturbed game. When the perturbations are decreased to zero in a series of perturbed games, the equilibria approximate an equilibrium in the original game that (may) differ from the subgame-perfect equilibrium: the so-called *trembling-hand equilibrium.* Unfortunately, there is no practical algorithm to compute this equilibrium for large game trees.

The idea of perturbed games has a close resemblance with the earlier work of Reibman and Ballard (1983). Their *-Minimax search method assumes that MIN has a probability to be fallible (see section 2.1.3). It is, however, not clear whether *-Minimax search would yield other results than Minimax when the fallibility of MIN approaches zero. Also the methods of Slagle and Dixon (1970) and Uiterwijk and Van den Herik (1994) can be related to the idea of the trembling-hand equilibrium.

For a detailed discussion on the many alternatives for subgame-perfect equilibria, we refer to chapter 8 of Fudenberg and Tirole (1991).

## 2.6 Concluding Remarks

We started this chapter with search methods that preceded the discovery of OM search. These methods all used some implicit model of the opponent's behaviour to improve the search result. OM search is a search method that utilizes an explicit opponent model. Since OM search depends on the exact prediction of the opponent's move, knowledge of the opponent is important and errors in the prediction can destroy the possible gains of OM search. In the experiments that both Carmel and Markovitch (1996a) and Gao *et al.* (2001) performed with OM search, perfect knowledge of the opponent was available. The influence of prediction errors on the performance of OM search is one of the topics that is investigated in this thesis, not only theoretically, but also in real game settings.

For the practical applicability of a search method, the efficiency is as important as the quality of the search result. Several implementations of OM search and M search have been developed, including $\beta$ pruning, but there is still room for improvement. In this thesis we present alternative implementations that can prune more of the game tree and can profit from search enhancements such as iterative deepening and transposition tables.

A more fundamental study is needed on the essentials of how OM search treats a game tree. From a game-theoretical point of view, OM search is solving an incomplete-information game: MAX has considerably more knowledge on the payoffs than MIN. The type of knowledge is such that a solution is easily defined: MAX knows MIN's complete strategy, so the task is to find the best strategy as an answer to this. It is a maximization task: find the strategy that, given the opponent's strategy, maximizes the own payoff. Maximizing is quite different from minimaxing, because maximizing is attracted to places in the tree with maximal payoffs. Minimaxing is somehow trying to find the mean of all possible payoffs. The maximizing behaviour of OM search has influence on how OM search responds to errors in MAX's evaluation function.

Perfect knowledge of the opponent is rare in practice. In this thesis we want to relax the perfect-predictability demand of OM search. In a new search method (PrOM search, for Probabilistic Opponent-Model search), we will introduce uncertainty in the opponent model. We will investigate whether PrOM search will elevate some of the problems of OM search.

In the last part of this chapter we showed that some of the search methods described are connected to the recent discussion on equilibria in game theory. This connection expresses a mutual interest which we formulate by the following two questions: (1) can search methods be developed that can efficiently find equilibria in other game types than the classical zero-sum games? and (2) can the definition of new equilibria be used to improve the quality of play by computers?

# Chapter 3

# Opponent-Model Search: Probing and Pruning

*At first glance, OM search seems a promising search method since it incorporates an opponent model. The question how this method can be applied in a practical setting turns out to be difficult. There are issues of efficiency and of effectiveness. In this chapter we deal with efficiency: the search method must be implemented efficiently. One approach is applying probes that use $\alpha$-$\beta$ search. However, a major factor of efficiency is the prevailing question how much of the tree can be pruned during search. For this purpose a full list of search enhancements is investigated.*

---

**Chapter contents:** In this chapter, the implementation of OM search is investigated. Plain OM search is treated in section 3.1; then pruning is added in section 3.2. A best-case analysis of the algorithms is performed in section 3.3. In section 3.4 the applicability of a list of search enhancements on OM search is investigated. Finally, section 3.5 provides the chapter conclusions.[1]

## 3.1 Implementing OM Search

The description of OM search in subsection 2.3.1 specified the relation between the values of parent nodes and child nodes, but it did not impose in which order

[1]Parts of this chapter have been published in Donkers, H.H.L.M., Uiterwijk, J.W.H.M., Herik, H.J. van den (2001), Probabilistic Opponent-Model Search, *Information Sciences*, Vol. 135, Nos. 3–4, pp. 123–149, and in Donkers, H.H.L.M., Uiterwijk, J.W.H.M., Herik, H.J. van den (2003), Admissibility in Opponent-Model Search, *Information Sciences*, Vol. 154, Nos. 3–4, pp. 119–140.

these values are determined. In this section we will study a number of possible implementations of OM seach.

### 3.1.1 Notation

Before we discuss the implementation of OM search, we first introduce our notation of the algorithms and some conventions. In algorithm 3.1 we present a version of $\alpha$-$\beta$ search as an example; it uses the symbols as introduced in section 1.2. As is clear from the head of the algorithm, we do not use a parameter for the search depth, but we use the set of terminal histories $E$ of the (reduced) game to indicate the leaf nodes of our search. In this way, we do not have to distinguish between true end positions and leaf nodes due to limited search depth. Furthermore, this notation allows for heterogeneous game trees. We use $V(\cdot)$ to indicate the passing of an evaluation function because in some of the algorithms below, we will call this algorithm with different evaluation functions from within an OM-search implementation. The set $E$ and the function $m(\cdot)$ are assumed to be globally available.

```
**AlphaBetaSearch**$(h, \alpha, \beta, V(\cdot))$
1   **if** $(h \in E)$ **return** $(V(h), \textbf{null})$
2   $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove$(L)$ ; $m^* \leftarrow m$
3   **if** $(p(h) = \textsc{max})$
4       $v^* \leftarrow \alpha$
5       **while** $(m \neq \textbf{null})$
6           $(v, mm) \leftarrow$ **AlphaBetaSearch**$(h + m, v^*, \beta, V(\cdot))$
7           **if** $(v > v^*)$ $v^* \leftarrow v$ ; $m^* \leftarrow m$
8           **if** $(v^* \geq \beta)$ $m \leftarrow$ **null else** $m \leftarrow$ nextMove$(L)$
9   **if** $(p(h) = \textsc{min})$
10      $v^* \leftarrow \beta$
11      **while** $(m \neq \textbf{null})$
12          $(v, mm) \leftarrow$ **AlphaBetaSearch**$(h + m, \alpha, v^*, V(\cdot))$
13          **if** $(v < v^*)$ $v^* \leftarrow v$ ; $m^* \leftarrow m$
14          **if** $(v^* \leq \alpha)$ $m \leftarrow$ **null else** $m \leftarrow$ nextMove$(L)$
15  **return** $(v^*, m^*)$
```

Algorithm 3.1: $\alpha$-$\beta$ search.

The symbol $L$ is used to indicate a list of moves and the two operations firstMove and nextMove indicate the retrieval of the first and subsequent moves from this list. When no move is available (anymore), the operations return **null**. The actual ordering of the moves is implicit in the assignment of $L$. The execution of moves is not explicitly notated, but is inherent in the notation $h + m$ of the recursive call at lines 6 and 12. The search algorithms that we discuss in this chapter can return more than one value at a time. The algorithm **AlphaBetaSearch** returns two values: first the game-tree value and second the best move. An asterisk is used to indicate optimal values or best choices. The player function $p(h)$ allows to distinguish between max nodes and min nodes.

We use indentation to express the flow of the algorithm: after an **if**, **else** or **while** statement, all indented statements are conditional. If there is more than one statement at one line after an **if** or **else** statement, these statements are all conditional.

### 3.1.2 Plain OM search

There are at least two ways in which the equations 2.10 and 2.11 can be implemented with respect to the traversal of the game tree. The first way is a one-pass algorithm that visits all nodes in the algorithm at most once and determines the best choice for MIN and MAX at the same time. This algorithm is called **OmSearch1p**. The second way is an algorithm that predicts the choices for MIN using a Minimax strategy and then selects the best choices for MAX. It is called **OmSearchPb**

Algorithm 3.2 (**OmSearch1p**) is a one-pass implementation of OM search. It returns three values: the game-tree values for MAX and for MIN and the best choice for MAX. The passing of both game-tree values together is necessary for the computation of the maxima at max nodes for MAX and MIN simultaneously (lines 7 and 8). The passing of both values is also necessary at min nodes in order to determine the value of $v_0$ for the child node that results in the minimum value of $v_{op}$. At line 11, $v_0$ is initialized with an arbitrary value (0). When this value of $v_0$ is wrongly used to set $v_0^*$ then this will be repaired at line 17.

The second straightforward way to implement OM search is by applying *probing*: at every min node perform Minimax search with the opponent's evaluation function

```
OmSearch1p($h$)
1   if ($h \in E$) return ($V_0(h), V_{op}(h)$, null)
2   $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$) ; $m^* \leftarrow m$
3   if ($p(h) =$ MAX)
4       $v_0^* \leftarrow -\infty$ ; $v_{op}^* \leftarrow -\infty$
5       while ($m \neq$ null)
6           $(v_0, v_{op}, mm) \leftarrow$ OmSearch1p($h + m$)
7           if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m^* \leftarrow m$
8           if ($v_{op} > v_{op}^*$) $v_{op}^* \leftarrow v_{op}$
9           $m \leftarrow$ nextMove($L$)
10  if ($p(h) =$ MIN)
11      $v_{op}^* \leftarrow +\infty$ ; $v_0 \leftarrow 0$
12      while ($m \neq$ null)
13          if ($h + m \in E$) $v_{op} \leftarrow V_{op}(h + m)$
14              else $(v_0, v_{op}, mm) \leftarrow$ OmSearch1p($h + m$)
15          if ($v_{op} < v_{op}^*$) $v_{op}^* \leftarrow v_{op}$ ; $v_0^* \leftarrow v_0$ ; $m^* \leftarrow m$
16          $m \leftarrow$ nextMove($L$)
17      if ($h + m^* \in E$) $v_0^* \leftarrow V_0(h + m^*)$
18  return ($v_0^*, v_{op}^*, m^*$)
```

Algorithm 3.2: One-pass OM search.

```
OmSearchPb($h$)
1   if ($h \in E$) return ($V_0(h)$, null)
2   if ($p(h)$ = MAX)
3       $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$)
4       $m^* \leftarrow m$ ; $v_0^* \leftarrow -\infty$
5       while ($m \neq$ null)
6           $(v_0, mm) \leftarrow$ OmSearchPb($h + m$)
7           if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m^* \leftarrow m$
8           $m \leftarrow$ nextMove($L$)
9   if ($p(h)$ = MIN)
10      $(v_{op}^*, m^*) \leftarrow$ AlphaBetaSearch($h, -\infty, \infty, V_{op}(\cdot)$)
11      $(v_0^*, mm) \leftarrow$ OmSearchPb($h + m^*$)
12  return ($v_0^*, m^*$)
```

Algorithm 3.3: OM search with $\alpha$-$\beta$ probing.

(the *probe*), and perform OM search at the position that is reached after the move that the probe returns is performed; at every max node, maximize over all child nodes. The probes can in fact be implemented using any enhanced Minimax search algorithm available. Because a separate probe is performed for every min node, many nodes are visited during multiple probes. Algorithm 3.3 (**OmSearchPb**) is an implementation of OM search using $\alpha$-$\beta$ probing (at line 10). Since there is no concurrent maximization of values for MAX and MIN, this version of OM search returns only two values: the game-tree value for MAX and the best choice for MAX.

The one-pass algorithm 3.2 (**OmSearch1p**) is equivalent to the $M^*_{1\text{-}pass}$ implementation of Carmel and Markovitch (1996a; 1998) in case of a one-level opponent model and a fixed search depth (see subsection 2.4.4). Algorithm 3.3 (**OmSearchPb**) is equivalent to the $M^*$ search implementation of Carmel and Markovitch augmented with *simple pruning* as described in section 3.1 of their (1998) publication (also in case of a one-level opponent model and fixed search depth).

### 3.1.3 Implementing the enhanced formulae of OM search

Both algorithms 3.2 (**OmSearch1p**) and 3.3 (**OmSearchPb**) are based on the formulae for OM search in subsection 2.3.1. In subsection 2.3.2 we gave an enhanced version of these formulae. The enhanced formulae deal with different search depths for MAX and MIN and with moves that are equally evaluated by MIN.

The handling of different search depths can be incorporated by changing line 1 in algorithms 3.2 and 3.3. (In general, we have marked changes with respect to main algorithms by printing the line numbers bold.) In the one-pass version of the enhanced algorithm (3.4), called **OmSearchEnh1p**, a node is evaluated if it is a leaf node for MAX or for MIN. If a node is a leaf node for both players at the same time, the algorithm returns the two evaluations ($V_0(h)$ and $V_{op}(h)$ immediately (line 1a). Otherwise, a call to **AlphaBetaSearch** is performed for the player who still has some search depth left (for MAX: line 1b; for MIN: line 1c). Because lines 13 and 17 in the original algorithm 3.2 (**OmSearch1p**) look one move ahead, these lines

```
OmSearchEnh1p($h$)
1a  if ($h \in E_0 \cap E_{op}$) return ($V_0(h), V_{op}(h)$, null)
1b  if ($h \in E_0$) ($v^*_{op}, m^*$) $\leftarrow$ AlphaBetaSearch($h, -\infty, \infty, V_{op}(\cdot)$) ;
                                                return ($V_0(h), v^*_{op}$, null)
1c  if ($h \in E_{op}$) ($v^*_0, m^*$) $\leftarrow$ AlphaBetaSearch($h, -\infty, \infty, V_0(\cdot)$) ;
                                                return ($v^*_0, V_{op}(h), m^*$)
2   $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$) ; $m^* \leftarrow m$
3   if ($p(h) =$ MAX)
4       $v^*_0 \leftarrow -\infty$ ; $v^*_{op} \leftarrow -\infty$
5       while ($m \neq$ null)
6           ($v_0, v_{op}, mm$) $\leftarrow$ OmSearchEnh1p($h + m$)
7           if ($v_0 > v^*_0$) $v^*_0 \leftarrow v_0$ ; $m^* \leftarrow m$
8           if ($v_{op} > v^*_{op}$) $v^*_{op} \leftarrow v_{op}$
9           $m \leftarrow$ nextMove($L$)
10  if ($p(h) =$ MIN)
11      $v^*_{op} \leftarrow +\infty$ ; $v_0 \leftarrow 0$
12      while ($m \neq$ null)
13          if ($h + m \in E_{op}$) $v_{op} \leftarrow V_{op}(h + m)$
14              else ($v_0, v_{op}, mm$) $\leftarrow$ OmSearchEnh1p($h + m$)
15          if ($v_{op} < v^*_{op}$) $v^*_{op} \leftarrow v_{op}$ ; $v^*_0 \leftarrow v_0$ ; $m^* \leftarrow m$
15a             else if ($v_{op} = v^*_{op}$ and $v_0 < v^*_0$) $v^*_0 \leftarrow v_0$ ; $m^* \leftarrow m$
16          $m \leftarrow$ nextMove($L$)
17          if ($h + m^* \in E_{op}$)
                    ($v^*_0, m'$) $\leftarrow$ AlphaBetaSearch($h + m^*, -\infty, \infty, V_0(\cdot)$)
18  return ($v^*_0, v^*_{op}, m^*$)
```

Algorithm 3.4: One-pass enhanced OM search.

are slightly adapted, too, in algorithm 3.4 (**OmSearchEnh1p**).

In the probing version of OM search, the adaptation to different search depths is different. Below we describe the implementation of enhanced OM search with $\alpha$-$\beta$ probing. It is called **OmSearchEnhPb**, see algorithm 3.5. At a leaf node for MAX, the enhanced algorithm returns MAX's evaluation only (line 1a); at nodes where MAX searches deeper than MIN (i.e., all histories outside $H_{op} \setminus E_{op}$; histories that are not internal nodes for MIN), the algorithm calls **AlphaBetaSearch** with MAX's evaluation function (line 1b). The cases in which MIN searches deeper than MAX are covered by the probes at line 10, under the condition that **AlphaBetaSearch** knows which set $E$ is to be used. Both algorithms 3.4 and 3.5 can also be used as an implementation of $(D, d)$-OM search (see subsection 2.4.3).

The algorithms are also easily adapted to cope with moves that are equally evaluated by MIN. Line 15a in the one-pass algorithm 3.4 (**OmSearchEnh1p**) deals with equal values of $v^*_{op}$: in such case it selects the move with the minimum value for $v_0$. The handling of equally evaluated moves in the probing version (algorithm 3.5: **OmSearchEnhPb**) requires an adaptation of the $\alpha$-$\beta$-search implementation: instead of returning one best choice, it should return a list of all best choices (indic-

```
**OmSearchEnhPb**($h$)
1a  if ($h \in E_0$) return ($V_0(h)$, null)
1b  if ($h \notin H_{op} \setminus E_{op}$) return AlphaBetaSearch($h, -\infty, \infty, V_0(\cdot)$)
2   if ($p(h) =$ MAX)
3       $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$)
4       $m^* \leftarrow m$ ; $v_0^* \leftarrow -\infty$
5       while ($m \neq$ null)
6           $(v_0, mm) \leftarrow$ OmSearchEnhPb($h + m$)
7           if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m^* \leftarrow m$
8           $m \leftarrow$ nextMove($L$)
9   if ($p(h) =$ MIN)
10      $(v_{op}^*, L) \leftarrow$ AlphaBetaSearch($h, -\infty, \infty, V_{op}(\cdot)$)
11      $m \leftarrow$ firstMove($L$) ; $m^* \leftarrow m$ ; $v_0^* \leftarrow +\infty$
12      while ($m \neq$ null)
13          $(v_0, mm) \leftarrow$ OmSearchEnhPb($h + m$)
14          if ($v_0 < v_0^*$) $v_0^* \leftarrow v_0$ ; $m^* \leftarrow m$
15          $m \leftarrow$ nextMove($L$)
16  return ($v_0^*, m^*$)
```

Algorithm 3.5: Enhanced OM search with $\alpha$-$\beta$ probing.

ated by $L$ in line 10). For all moves on this list, OM search must be called in order to determine the minimum value for these moves for MAX (lines 11 to 15).

The algorithms in the rest of this chapter are based on the plain implementations of OM search (algorithms 3.2 and 3.3). The adaptations that are introduced in algorithms 3.4 and 3.5 can easily be added to the algorithms below, so it is not needed to mention them each time. It keeps the algorithms smaller and more clear.

### 3.1.4 Restricted speculation

OM search assumes that MAX speculates at 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 probes than in the higher parts of the tree. It could therefore be justified to speculate only in the upper portion of the search tree. Algorithm 3.6, called **OmSearchRsPb**, is an adaptation of OM search with $\alpha$-$\beta$ probing in which speculation is restricted to the histories in $H_{spec} \subseteq H$. The *speculative search depth* is defined as the (maximum) length of the histories in $H_{spec}$. We provide a version of restricted speculation with $\alpha$-$\beta$ probing only. The one-pass version is a similar straightforward adaptation of algorithm 3.2. The most restricted speculation occurs when $H_{spec}$ is empty, since in that case only $\alpha$-$\beta$ search is performed. A more realistic setting is the restriction of $H_{spec}$ to all histories $h \in H$ that do not exceed length 2, resulting in speculation only at the upmost min nodes in the tree.

```
OmSearchRsPb($h$)
1   if ($h \notin H_{spec}$) return AlphaBetaSearch($h, -\infty, \infty, V_0(\cdot)$)
2   if ($p(h) = $ MAX)
3       $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$)
4       $m^* \leftarrow m$ ; $v_0^* \leftarrow -\infty$
5       while ($m \neq$ null)
6           $(v_0, mm) \leftarrow$ OmSearchRsPb($h + m$)
7           if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m^* \leftarrow m$
8           $m \leftarrow$ nextMove($L$)
9   if ($p(h) = $ MIN)
10      $(v_{op}^*, m^*) \leftarrow$ AlphaBetaSearch($h, -\infty, \infty, V_{op}(\cdot)$)
11      $(v_0^*, mm) \leftarrow$ OmSearchRsPb($h + m^*$)
12  return $(v_0^*, m^*)$
```

Algorithm 3.6: OM search with $\alpha$-$\beta$ probing and restricted speculation.

## 3.2 Pruning in OM Search

The success of $\alpha$-$\beta$ search is mainly caused by the large power of this algorithm to prune the search tree (Knuth and Moore, 1975). If OM search is ever to be practically applicable, it must be able to apply pruning to the search tree as well. In this section we will discuss two pruning mechanisms that have been developed for OM search: $\beta$ pruning and bounded-sum pruning. We will provide a one-pass version and a version with $\alpha$-$\beta$ probing for both pruning mechanisms. In section 3.3 we will analyze the best-case efficiency of the two $\beta$-pruning versions.

### 3.2.1 $\beta$-pruning OM search

*$\beta$-Pruning OM search* (Iida *et al.*, 1993a) is an improvement of plain OM search; it has a better efficiency and yields the same result. The method prunes the search tree in max nodes (not in min nodes, hence the name '$\beta$-pruning'). The pruning in this method is analogous to the pruning in $\alpha$-$\beta$ search: when for one of the children $m_j$ of a non-leaf max node $h$ the value of $v_{op}(h + m_j)$ is higher than for the already evaluated siblings of the max node, then the remaining children of the max node can be pruned. At node ***e*** in figure 3.1, for example, the inspection of further child nodes can be omitted after the child ***j*** returned a value greater than the current value of $\beta$. The same happens at node ***g*** after the evaluation of node ***m***. It is a safe pruning since it is certain that such a max node $h$ will never be chosen at its parent min node and so its value cannot influence the value of the root. This means that the precise values for $v_0(h)$ and $v_{op}(h)$ are not needed. Pruning at internal min nodes $h$ is not possible, because the value of $v_0(h)$ depends on the exact minimum of $v_{op}(h + m_j)$ over moves $m_j \in m(h)$.

$\beta$-Pruning OM search also discards unnecessary evaluations of $V_0(\cdot)$ at leaf nodes. There are two cases to distinguish. The first case concerns leaf nodes of the search tree that are max nodes. In this case $V_0$ only has to be determined for those leaf nodes that have the lowest $V_{op}$ among their siblings (nodes ***o***, ***p***, ***t***, and ***w*** in figure

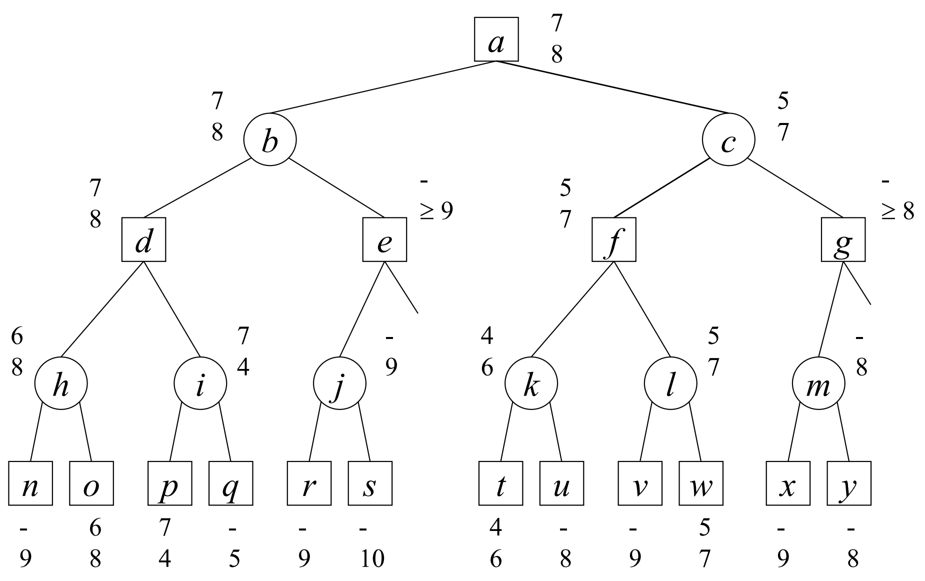


Figure 3.1: A $\beta$-pruning OM-search example. The upper numbers next to the nodes (and under the leafs) indicate $v_0$, the lower numbers indicate $v_{op}$. A dash means that a value need not to be determined.

3.1). When the value of the lowest $V_{op}$ among siblings happens to be higher than the current value of $\beta$, the value of $V_0$ does not have to be determined at all for this node (node ***r*** in figure 3.1).

The second case concerns leaf nodes that are min nodes. The value of $V_0$ only has to be determined for those min leaf nodes that have a value of $V_{op}$ that does not exceed the current value of $\beta$ (node ***m*** in figure 3.1).

Depth-first $\beta$-pruning OM search can be implemented both in a one-pass version and in a version with $\alpha$-$\beta$ probing as enhancements of the non-pruning versions of OM search (algorithms 3.2 and 3.3).

### 3.2.2 One-pass $\beta$-pruning OM search

Dealing with max nodes is easy since one-pass $\beta$-pruning OM search can simply be called recursively for all child nodes from left to right until the value of $\beta$ is exceeded, see figure 3.1.

Dealing with min nodes is more elaborate. When a min node $h$ has only child nodes that are leaf nodes then it is easy to find the move $m_j$ that leads to the lowest value of $v_{op}(h + m_j)$ because only the value of $V_{op}(\cdot)$ has to be computed for every child node. The value of $V_0(\cdot)$ needs to be computed only for the child node with the lowest value of $V_{op}(\cdot)$, provided that the lowest value of $V_{op}(\cdot)$ is smaller than the value of $\beta$ that was passed to the node. If the lowest value of $V_{op}(\cdot)$ is equal to or smaller than the $\beta$ no $V_0(\cdot)$ value has to be computed at all.

When the child nodes of a min node are internal nodes, it is more difficult to find

```
**OmSearchBeta1p**$(h, \beta)$
1  **if** $(h \in E)$ **return** $(V_0(h), V_{op}(h), \textbf{null})$
2  $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove$(L)$ ; $m^* \leftarrow m$
3  **if** $(p(h) = \text{MAX})$
4      $v_0^* \leftarrow -\infty$ ; $v_{op}^* \leftarrow -\infty$
5      **while** $(m \neq \textbf{null})$
6          **if** $(h + m \in E)$ $v_{op} \leftarrow V_{op}(h+m)$ ; **if** $(v_{op} < \beta)$ $v_0 \leftarrow V_0(h+m)$
7              **else** $(v_0, v_{op}, mm) \leftarrow$ **OmSearchBeta1p**$(h+m, \beta)$
8          **if** $(v_0 > v_0^*)$ $v_0^* \leftarrow v_0$ ; $m^* \leftarrow m$
9          **if** $(v_{op} > v_{op}^*)$ $v_{op}^* \leftarrow v_{op}$
10         **if** $(v_{op}^* \geq \beta)$ $m \leftarrow$ **null else** $m \leftarrow$ nextMove$(L)$
11 **if** $(p(h) = \text{MIN})$
12     $v_{op}^* \leftarrow \beta$ ; $v_0 \leftarrow 0$
13     **while** $(m \neq \textbf{null})$
14         **if** $(h + m \in E)$ $v_{op} \leftarrow V_{op}(h+m)$
15             **else** $(v_0, v_{op}, mm) \leftarrow$ **OmSearchBeta1p**$(h+m, v_{op}^*)$
16         **if** $(v_{op} < v_{op}^*)$ $v_{op}^* \leftarrow v_{op}$ ; $v_0^* \leftarrow v_0$ ; $m^* \leftarrow m$
17         $m \leftarrow$ nextMove$(L)$
18     **if** $(h + m^* \in E \ \wedge \ v_{op}^* < \beta)$ $v_0^* \leftarrow V_0(h+m^*)$
19 **return** $(v_0^*, v_{op}^*, m^*)$
```

Algorithm 3.7: One-pass $\beta$-pruning OM search.

the move with the lowest value of $v_{op}(\cdot)$. The one-pass version applies $\beta$-pruning OM search recursively to all child nodes $h + m_j$ of $h$. This results in the simultaneous computation of two values, viz. that of $v_0(h + m_j)$ and of $v_{op}(h + m_j)$, for every child node. So, the value of $v_0(h)$ is found as soon as the minimum value for $v_{op}(\cdot)$ is detected. The value of $\beta$ that the recursive call passes is either the value of $\beta$ that was received from the parent node, or the largest value of $v_{op}(\cdot)$ if this value is greater than the current $\beta$ parameter. This is equal to handling the $\beta$ parameter in **AlphaBetaSearch** (algorithm 3.1). We call the one-pass version of $\beta$ pruning OM search **OmSearchBeta1p** and abbreviate this name in the text to OM$^{\beta 1p}$.

The implementation of OM$^{\beta 1p}$ is given in algorithm 3.7. The $\beta$-pruning at leaf nodes takes place in the lines 6 and 18. The $\beta$-pruning at internal max nodes takes place in line 10: by setting $m$ to **null**, the iteration of child nodes is terminated and the remaining child nodes are pruned.

Finally we remark that the value of $v_{op}(h + m_j)$ is needed for all $m_j$, and that the value of $v_0(h + m_j)$ is only needed for that move $m_j$ that has the lowest value for $v_{op}(h + m_j)$. This means that values of $V_0(\cdot)$ will thus be computed too often in the one-pass version.

### 3.2.3 $\beta$-pruning OM search with $\alpha$-$\beta$ probes

It is also possible to apply $\beta$ pruning to the version with $\alpha$-$\beta$ probes. When an $\alpha$-$\beta$ probe with a $[-\infty, +\infty]$ window is applied to a min node, as in line 10 of

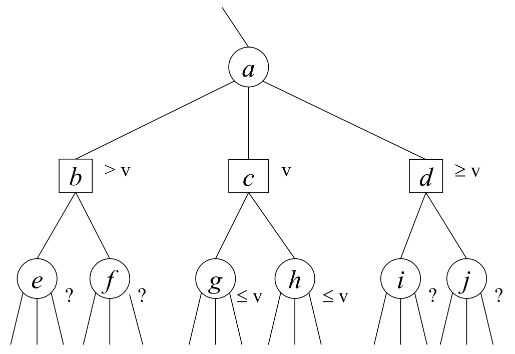


Figure 3.2: An example to illustrate the $\beta$ parameter of $\alpha$-$\beta$ probes.

algorithm 3.3, then $\beta$ pruning below max nodes already takes place automatically. However, the $\alpha$-$\beta$ probes at a min node $h$ and at its grandchild min nodes $h+m_j+m_k$ are not independent because the $\alpha$-$\beta$ value of $h$ ($v_{op}(h)$) is necessarily equal or greater than all $\alpha$-$\beta$ values of $h+m_j+m_k$. This means that $v_{op}(h)$ can be used to reduce the window of the probes at the grandchild nodes.

Figure 3.2 illustrates how the $\beta$ parameters for the probes can be set. Assume that an $\alpha$-$\beta$ probe at node ***a*** has returned value $v$ and the move leading to node ***c***. The following four conclusions can be derived from this knowledge:

(1) the subgame value for node ***b*** is greater than $v$,
(2) the subgame value for node ***d*** is greater than or equal to $v$,
(3) no bound can be given for the subgame values of nodes ***e***, ***f***, ***i***, and ***j***,
(4) the subgame values for nodes ***g*** and ***h*** are less than or equal to $v$.

This knowledge has consequences for nodes ***g*** and ***h***: a window of $[-\infty, v+1]$ is allowed for the $\alpha$-$\beta$ probes at these nodes. The $\beta$ parameter must be set to $v+1$ in

```
OmSearchBetaPb($h, \beta$)
1   if ($h \in E$) return ($V_0(h)$, null)
2   if ($p(h) =$ MAX)
3       $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$)
4       $m^* \leftarrow m$ ; $v_0^* \leftarrow -\infty$
5       while ($m \neq$ null)
6           ($v_0, mm$) $\leftarrow$ OmSearchBetaPb($h+m, \beta$)
7           if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m^* \leftarrow m$
8           $m \leftarrow$ nextMove($L$)
9   if ($p(h) =$ MIN)
10      ($v_{op}^*, m^*$) $\leftarrow$ AlphaBetaSearch($h, -\infty, \beta, V_{op}(\cdot)$)
11      ($v_0^*, mm$) $\leftarrow$ OmSearchBetaPb($h+m^*, v_{op}^*+1$)
12  return ($v_0^*, m^*$)
```

Algorithm 3.8: $\beta$-pruning OM search with $\alpha$-$\beta$ probing.

order to prevent premature cuts in the $\alpha$-$\beta$ probe in the case that the value of the node is equal to $v$. These premature cuts can cause $\alpha$-$\beta$ probes to select the wrong move as optimal.

This enhancement of OM search was called *$\beta$-passing* in Donkers, Uiterwijk, and Van den Herik (2001). In this thesis we will speak about $\beta$-pruning OM search with $\alpha$-$\beta$ probes. The algorithm that implements this enhancement is called **OmSearch-BetaPb**, which we will abbreviate to $\text{OM}^{\beta Pb}$ for convenience. The implementation of $\text{OM}^{\beta Pb}$ is given in algorithm 3.8.

### 3.2.4 Bounded-sum pruning

Carmel and Markovitch (1996a; 1998) developed another pruning method for M* search, based on a bounded difference between the opponent models (introduced in 1996a, improved in 1998). Since OM search is a special case of M* search, this pruning method is evidently applicable to OM search as well. The absence of a recursive opponent model in OM search causes the pruning to be less complex in OM search than in the general case of M* search. We will call the pruning method *bounded-sum pruning* to indicate the preconditions for the application of this method.[2]

The fundament for bounded-sum pruning is the observation that if the absolute difference between $V_0(h)$ and $V_{op}(h)$ is bounded to some positive limit $B$ for all $h \in E$, then the absolute difference between values $v_0(h)$ and $v_{op}(h)$ is also bounded to $B$ for all $h \in H$. This means that the value of $v_{op}(h)$ can be used to give a bound for the value of $v_0(h)$ before it is determined, possibly resulting in pruning at max nodes.

The truth of the observation is easily proven by induction; the reasoning is as follows.[3] The observation is clearly true for all $h \in E$. Assume that for a given $h$, $|v_0(h+m) - v_{op}(h+m)| \leq B$ for all $m \in m(h)$. Further, assume that $v_0(h) = v_0(h+m_i)$ and $v_{op}(h) = v_{op}(h+m_j)$ (MAX selects $m_i$ and MIN selects $m_j$). If $i = j$, which is always true when $p(h)$=MIN, then $v_0(h)$ and $v_{op}(h)$ were propagated from the same node and therefore their difference is bounded by $B$. If $p(h)$=MAX and $i \neq j$, it must hold that $v_0(h+m_j) \leq v_0(h+m_i)$ and $v_{op}(h+m_i) \leq v_{op}(h+m_j)$; otherwise moves $m_i$ and $m_j$ would not have been selected. There are two different cases (see figure 3.3). The first case is the crossed configuration in which $v_0(h+m_i) \geq v_{op}(h+m_i)$ and $v_{op}(h+m_j) \geq v_0(h+m_j)$. The value of $v_{op}(h+m_j)$ cannot be smaller than $v_{op}(h+m_i)$, so $v_{op}(h+m_j)$ cannot lie more below $v_0(h+m_i)$ than $B$. Likewise, $v_0(h+m_j)$ can never be greater than $v_0(h+m_i)$, so $v_{op}(h+m_j)$ can never lie more above $v_0(h+m_i)$ than $B$. The second case is the parallel configuration in which either $v_0$ or $v_{op}$ is maximal at both nodes. In this type of configuration, one of the intervals encloses the other: if $v_{op}$ is maximal, then the interval at $m_j$ encloses the interval at $m_i$ (as in the figure), if $v_0$ is maximal the interval at $m_j$ is enclosed. In

[2]We prefer the term *bounded-sum* as analogue to *zero-sum games*, although it is in fact the difference between evaluation functions $V_0$ and $V_{op}$ that has to be bounded. The preference is caused by the way how we deal with both $V_0$ and $V_{op}$, viz. from the viewpoint of MAX. In a standard description of a non-zerosum game, $V_0$ and $V_{op}$ would have opposite viewpoints, so their sum should be bounded.

[3]Extension of the proof in Carmel and Markovitch (1998), where the case of a one-level model is not discussed.

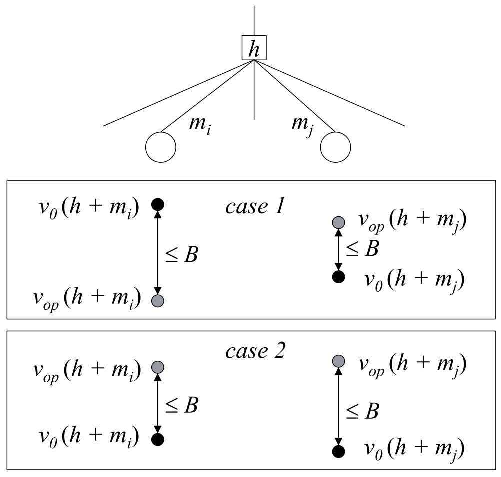


Figure 3.3: Example to support the proof of the bounded-sum property of OM-search trees.

both situations it is clear that the difference between $v_0(h + m_i)$ and $v_{op}(h + m_j)$ never can exceed $B$. All together, it implies that the difference between $v_0(h)$ and $v_{op}(h)$ is bounded to $B$ in all cases, which concludes the proof.

The one-pass version of the bounded-sum pruning algorithm for $\mathrm{M}^*_{1\text{-}pass}$ is called $\alpha$-$\beta^*_{1\text{-}pass}$. Carmel and Markovitch prove that this pruning algorithm is optimal in the sense that any one-pass algorithm that wants to prove the value of a game tree using the recursive opponent model, must evaluate all nodes that $\alpha$-$\beta^*_{1\text{-}pass}$ does. Of course, $\alpha$-$\beta^*_{1\text{-}pass}$ is also applicable to OM search. Algorithm 3.9 gives a translation of $\alpha$-$\beta^*_{1\text{-}pass}$ to OM search. The algorithm is called **OmSearchBounded1p** and uses separate $\alpha$'s and $\beta$'s for MIN and MAX. In lines 5 and 10, the search window for MAX is enlarged with two times $B$ because the (exact) value for MIN is only an approximate prediction for the value for MAX.

Algorithm 3.10 gives the version of bounded-sum pruning with $\alpha$-$\beta$ probing. It is called **OmSearchBoundedPb** and the code is a direct translation of the $\alpha$-$\beta^*$ algorithm by Carmel and Markovitch (1998). Pruning takes place at three places: at lines 8 and 13, and by the adaptations to the parameters of the $\alpha$-$\beta$ probe at line 10.

Bounded-sum pruning is only possible when both players have the same search depth ($E_0 = E_{op}$); otherwise it is not guaranteed that $v_0(h)$ and $v_{op}(h)$ are bounded to each other for all $h \in H$. If the bounded-sum property does hold, then this property has an interesting consequence because the proof above can easily be extended to the case in which MIN and MAX are *both* using Minimax search. In that case, the differences between the values for MAX and MIN ($v_{op}(h)$) are bounded to $B$, too, for all $h \in H$, including the root. The values of $v_0$ and $v_{op}$ at the root of the search tree are already bounded to $B$. This means that the maximum gain that can be reached by using OM search is bounded to $2B$: one time the maximum distance between the minimax value for MAX and $v_{op}$ and one time the maximum distance between $v_{op}$ and $v_0$.

```
$\textbf{OmSearchBounded1p}(h, \alpha_0, \alpha_{op}, \beta_0, \beta_{op})$
1   $\textbf{if}\ (h \in E)\ \textbf{return}\ (V_0(h), V_{op}(h), \textbf{null})$
2   $L \leftarrow m(h)\ ;\ m \leftarrow \text{firstMove}(L)\ ;\ m^* \leftarrow m$
3   $\textbf{if}\ (p(h) = \text{MAX})$
4       $v_0^* \leftarrow \alpha_0\ ;\ v_{op}^* \leftarrow \alpha_0\ ;\ \beta \leftarrow \beta_0$
5       $\textbf{if}\ (\alpha_0 < \beta_0)\ v_{op}^* \leftarrow \alpha_0 - B\ ;\ \beta \leftarrow \beta_0 + B$
6       $\textbf{while}\ (m \neq \textbf{null})$
7           $\textbf{if}\ (h + m \in E)\ v_{op} \leftarrow V_{op}(h + m)\ ;\ \textbf{if}\ (v_{op} \leq \beta_{op})\ v_0 \leftarrow V_0(h + m)$
8               $\textbf{else}\ (v_0, v_{op}, mm) \leftarrow \textbf{OmSearchBounded1p}(h + m, v_0^*,$
                                                   $v_{op}^* - B, \beta_{op} + B, \beta)$
9           $\textbf{if}\ (v_0 > v_0^*)\ v_0^* \leftarrow v_0\ ;\ m^* \leftarrow m$
10          $\textbf{if}\ (v_{op} > v_{op}^*)\ v_{op}^* \leftarrow v_{op}$
11          $\textbf{if}\ (v_{op}^* \geq \beta_{op})\ m \leftarrow \textbf{null}\ \textbf{else}\ m \leftarrow \text{nextMove}(L)$
12  $\textbf{if}\ (p(h) = \text{MIN})$
13      $v_{op}^* \leftarrow \beta_0\ ;\ \alpha \leftarrow \alpha_0\ ;\ v_0 \leftarrow 0$
14      $\textbf{if}\ (\alpha_0 < \beta_0)\ v_{op}^* \leftarrow \beta_0 + B\ ;\ \alpha \leftarrow \alpha_0 - B$
15      $\textbf{while}\ (m \neq \textbf{null})$
16          $\textbf{if}\ (h + m \in E)\ v_{op} \leftarrow V_{op}(h + m)$
17              $\textbf{else}\ (v_0, v_{op}, mm) \leftarrow$
                    $\textbf{OmSearchBounded1p}(h + m, \alpha_{op} - B, \alpha, \beta_0, v_{op}^*)$
18          $\textbf{if}\ (v_{op} < v_{op}^*)\ v_{op}^* \leftarrow v_{op}\ ;\ v_0^* \leftarrow v_0\ ;\ m^* \leftarrow m$
19          $\textbf{if}\ (v_{op}^* \leq \alpha_{op})\ m \leftarrow \textbf{null}\ \textbf{else}\ m \leftarrow \text{nextMove}(L)$
20      $\textbf{if}\ (h + m^* \in E\ \wedge\ v_{op}^* < \beta_{op})\ v_0^* \leftarrow V_0(h + m^*)$
21  $\textbf{return}\ (v_0^*, v_{op}^*, m^*)$
```

Algorithm 3.9: One-pass bounded-sum pruning OM search.

```
$\textbf{OmSearchBoundedPb}(h, \alpha, \beta)$
1   $\textbf{if}\ (h \in E)\ \textbf{return}\ (V_0(h), \textbf{null})$
2   $\textbf{if}\ (p(h) = \text{MAX})$
3       $L \leftarrow m(h)\ ;\ m \leftarrow \text{firstMove}(L)$
4       $m^* \leftarrow m\ ;\ v_0^* \leftarrow \alpha$
5       $\textbf{while}\ (m \neq \textbf{null})$
6           $(v_0, mm) \leftarrow \textbf{OmSearchBoundedPb}(h + m^*, \alpha, \beta)$
7           $\textbf{if}\ (v_0 > v_0^*)\ v_0^* \leftarrow v_0\ ;\ m^* \leftarrow m\ ;\ \textbf{if}\ (v_0^* > \alpha)\ \alpha \leftarrow v_0^*$
8           $\textbf{if}\ (\alpha \geq \beta)\ m \leftarrow \textbf{null}\ ;\ \textbf{else}\ m \leftarrow \text{nextMove}(L)$
9   $\textbf{if}\ (p(h) = \text{MIN})$
10      $(v_{op}^*, m^*) \leftarrow \textbf{AlphaBetaSearch}(h, \alpha - B, \beta + B, V_{op}(\cdot))$
11      $\textbf{if}\ (v_{op}^* - B > \alpha)\ \alpha \leftarrow v_{op}^* - B$
12      $\textbf{if}\ (v_{op}^* + B < \beta)\ \beta \leftarrow v_{op}^* + B$
13      $\textbf{if}\ (\alpha < \beta)\ (v_0^*, mm) \leftarrow \textbf{OmSearchBoundedPb}(h + m^*, \alpha, \beta)$
14          $\textbf{else}\ v_0^* \leftarrow \alpha$
15  $\textbf{return}\ (v_0^*, m^*)$
```

Algorithm 3.10: Bounded-sum pruning OM search with $\alpha$-$\beta$ probing.

## 3.3 Best-case analyses

In this section, the best-case behaviour of the two variants of $\beta$-pruning OM search is compared to $\alpha$-$\beta$ with respect to the time complexity. This behaviour is described for uniform game trees only. A uniform tree is a tree in which all internal nodes have the same number of children ($w$) and in which every leaf node is at the same depth ($d$). The game trees considered are assumed to contain no terminal positions.

The best-case input for a search algorithm can be defined in different ways. For $\alpha$-$\beta$-type search algorithms, it is normally defined as that specific ordering of a given tree among all possible orderings for which the algorithm needs to perform the minimal number of leaf-node evaluations. For algorithms like $\beta$-pruning OM search, such ordering is more difficult to define. Therefore the best-case input is defined here more generally as that ordered game tree among all possible ordered game trees of given size for which the algorithm performs the minimal number of leaf-node evaluations.

For $\alpha$-$\beta$ search, the two definitions coincide because a uniform game tree in which the nodes are optimally ordered, causes optimal pruning in $\alpha$-$\beta$ search. The number of leaf nodes from the total of $w^d$ in a uniform search tree of branching factor $w$ and depth $d$ that are evaluated reduces to:

$$C_{\alpha-\beta}(d,w) = w^{\lfloor d/2 \rfloor} + w^{\lceil d/2 \rceil} - 1 \tag{3.1}$$

if the tree is optimally ordered (see Knuth and Moore, 1975).

Below we will first discuss the best-case behaviour of the one-pass version of $\beta$-pruning OM search, followed by the best-case behaviour of the version with $\alpha$-$\beta$ probes. We do not discuss the best-case analysis for bounded-sum pruning, since its results depend heavily on the bound $B$ in relation to the detailed ranking of both evaluation functions. There are two limit cases for $B$: if $B = 0$ then bounded-sum pruning is equal to $\alpha$-$\beta$ search. If $B = +\infty$ then bounded-sum pruning is equal to $\beta$-pruning OM search and best-case analysis will follow below. Also, the enhanced versions of OM search and the restricted-speculation version will not be analysed separately.

### 3.3.1 Best-case analysis of OM$^{\beta 1p}$

In OM$^{\beta 1p}$, the effect of $\beta$-pruning is only dependent on the ordering of child nodes with respect to $v_{op}(\cdot)$. In uniform trees, the pruning is maximal if the tree is *well ordered*, which means that the child nodes $h_j$ of all nodes $h$ are sorted on their value of $v_{op}(h_j)$. For max nodes the child nodes must be ordered in decreasing order and for min nodes in increasing order.

**Theorem 1.** *Any algorithm that implements $\beta$-pruning OM search needs at least the number of evaluations on a* well-ordered uniform tree *that OM$^{\beta 1p}$ needs.*

*Proof.* To prove this theorem, we first determine this minimal number and then show that one-pass $\beta$-pruning OM search uses exactly the same number of evaluations.

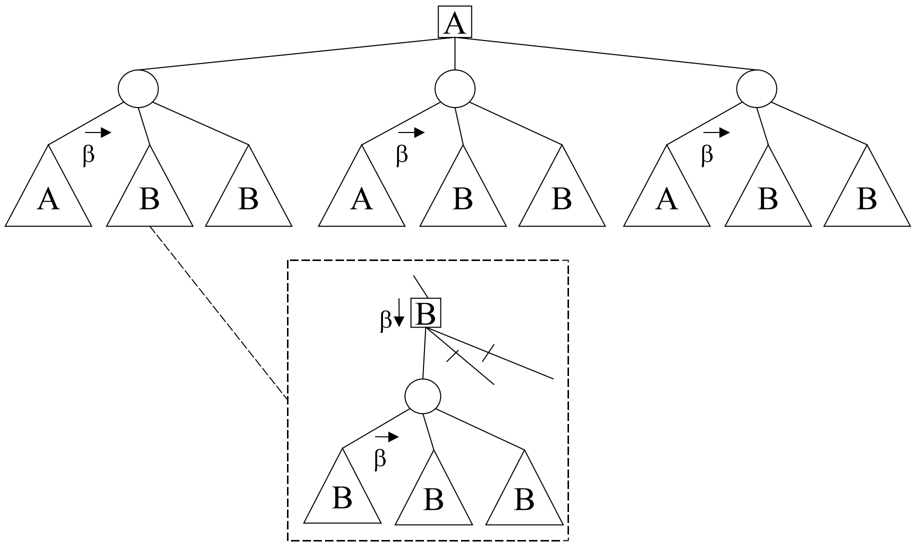


Figure 3.4: Theoretically best-case tree for $\beta$-pruning OM search.

Figure 3.4 gives a schematic representation of a well-ordered uniform tree to which $\beta$-pruning OM search is applied. From this figure we will derive a formula in closed form for the number of evaluations at least needed for $\beta$-pruning OM search.

Assume a uniform tree with branching factor $w$ and depth $d$. There are $w^2$ nodes at depth 2 to be expanded in such a search tree, because only branches at max nodes can be pruned and no pruning takes place at the root. The subtrees at these max nodes are of two different types. The first type of subtrees (type A) are the subtrees on the left-most branch of every node at depth 1. These subtrees are of the same type as the original tree and their number is $w$. The other $w(w-1)$ subtrees (type B) only have to be considered for the opponent's evaluation using $\alpha$-$\beta$ search with window $[-\infty, \beta]$. In figure 3.4, one of the type-B subtrees is worked out in detail. In this subtree, $\beta$ pruning takes place directly under the root, so only one min node remains at the first level. Directly below this min node, no $\alpha$ pruning can take place. The subtree at the first max child node of this min node is again of type B. The subtrees at the other max child nodes also are of type B. It is not possible that $\alpha$ pruning takes place in these subtrees because the maximal $\beta$ pruning at all max nodes prohibits the passing-through of $\alpha$ values.

The minimum number of evaluations $C_{OM}$ in the best case for any implementation of $\beta$-pruning OM search can now be given in the following recursive expression:

$$\begin{aligned} C_{OM}(d,w) &= w\,C_{OM}(d-2,w) + w(w-1)\,C'_{OM}(d-2,w) \\ C_{OM}(1,w) &= 2w \\ C_{OM}(2,w) &= w(w+1) \end{aligned} \tag{3.2}$$

In these formulae, $C_{OM}(d,w)$ stands for the number of evaluations needed at a subtree of type A, and $C'_{OM}(d,w)$ for a subtree of type B. In the case that the type-A subtree has depth 1, $2w$ evaluations are needed: $v_0(.)$ and $v_{op}(.)$ have to be evaluated for all $w$ leaf min nodes. When it has depth 2, $w(w+1)$ evaluations are needed: for all $w^2$ leaf max nodes, $v_{op}(.)$ has to be obtained, but only for the most left max node at every min child node, $v_0(.)$ is needed. The value of $C'_{OM}(d,w)$ is given by the next recursive expression:

$$\begin{aligned} C'_{OM}(d,w) &= w\, C'_{OM}(d-2,w) \\ C'_{OM}(1,w) &= 1 \\ C'_{OM}(2,w) &= w \end{aligned} \tag{3.3}$$

In the case that the type-B subtree has depth 1, only 1 evaluation is needed in the best case, since the value of $v_{op}(.)$ for the first child will be greater than $\beta$ and $v_0(.)$ does not have to be obtained. If the type-B subtree has depth 2, $w$ evaluations are needed, one for every grandchild of the first child node. Since all values for $v_{op}(.)$ will be greater than $\beta$, no value of $v_0(.)$ has to be obtained. Formula 3.3 can easily be written in closed form:

$$C'_{OM}(d,w) = w^{\lfloor d/2 \rfloor} \tag{3.4}$$

The equation for $C_{OM}$ can also be written in closed form (which can be found by applying repeated substitution):

$$\begin{gathered} C_{OM}(d,w) = k\, w^{\lceil d/2 \rceil} + (w-1) \sum_{i=1}^{\lceil d/2 \rceil - 1} w^i\, C'_{OM}(d-2i,w) \\ (k = 2 \text{ if } d \text{ is odd, } k = w+1 \text{ if } d \text{ is even}) \end{gathered} \tag{3.5}$$

The validity of the closed form can be proven by complete induction on $d$. For $d = 1$ and $d = 2$ equation (3.5) is clearly correct: the summation on the right hand side is zero in both cases. For $d > 2$ we first write down equation (3.5) with parameter $d-2$:

$$\begin{aligned} &C_{OM}(d-2,w) \\ &\quad = k\, w^{\lceil (d-2)/2 \rceil} + (w-1) \sum_{i=1}^{\lceil (d-2)/2 \rceil - 1} w^i\, C'_{OM}(d-2-2i,w) \\ &\quad = k\, w^{\lceil d/2 \rceil - 1} + (w-1) \sum_{i=1}^{\lceil d/2 \rceil - 2} w^i\, C'_{OM}(d-2-2i,w) \\ &\quad = k\, w^{\lceil d/2 \rceil - 1} + (w-1) \sum_{i=2}^{\lceil d/2 \rceil - 1} w^{i-1}\, C'_{OM}(d-2i,w) \end{aligned} \tag{3.6}$$

Substituting equation (3.6) into (3.2) results directly in equation (3.5), which proves the correctness of the closed form. The closed form of equation (3.5) can be reduced further by applying equation (3.4) and canceling out the summation:

$$\begin{aligned} C_{OM}(d,w) &= k\,w^{\lceil d/2 \rceil} + (w-1) \sum_{i=1}^{\lceil d/2 \rceil -1} w^i w^{\lfloor (d-2i)/2 \rfloor} \\ &= k\,w^{\lceil d/2 \rceil} + (w-1) \sum_{i=1}^{\lceil d/2 \rceil -1} w^{\lfloor d/2 \rfloor} \end{aligned} \tag{3.7}$$

which can be reduced to:

$$C_{OM}(d,w) = k\,w^{\lceil d/2 \rceil} + (w-1)\,(\lceil d/2 \rceil - 1)\,w^{\lfloor d/2 \rfloor} \tag{3.8}$$

The expression $k\,w^{\lceil d/2 \rceil}$ can be rewritten to $w^{\lfloor d/2 \rfloor + 1} + w^{\lceil d/2 \rceil}$, which removes the $k$. This can be used to rewrite the equation to:

$$C_{OM}(d,w) = w^{\lceil d/2 \rceil} + w^{\lfloor d/2 \rfloor} + (w-1)\,\lceil d/2 \rceil\,w^{\lfloor d/2 \rfloor} \tag{3.9}$$

This concludes the first part of the proof. The number of evaluations in the best case for OM$^{\beta 1p}$ appears to be equal to $C_{OM}(d,w)$. The reasoning is as follows (figure 3.4 can be used to illustrate this result). The type-A subtrees are of the same type as the original tree, just like the theoretical case. This means that the overall formula 3.5 also holds for OM$^{\beta 1p}$. However, OM$^{\beta 1p}$ does not apply $\alpha$-$\beta$ search to the type-B subtrees. Fortunately, in the best case optimal $\beta$ pruning on all internal max nodes and on all leaf nodes can take place. So no evaluation of $V_0$ takes place at all in type-B subtrees and the number of evaluations in these type-B subtrees is given by equation 3.4. Now the theoretical derivation van be followed. This concludes the proof. □

### 3.3.2 Best-case analysis of OM$^{\beta Pb}$

In $\beta$-pruning OM search with $\alpha$-$\beta$ probes, the number of leaf nodes that are evaluated for MAX's evaluation function depends only on the size of the tree, not on the ordering of the nodes. For the moment, the $\alpha$-$\beta$ probes can be disregarded. At every max node, all $w$ child nodes are visited and at every min node, exactly 1 child is visited. This means that there are exactly $w^{\lceil d/2 \rceil}$ leaf nodes visited and evaluated for MAX.

The number of $\alpha$-$\beta$ probes in OM search is also dependent only on the size of the tree. At every odd ply $2i-1$ ($i > 0$), exactly $w^i$ probes are performed. The $\alpha$-$\beta$ probes at the first odd ply have a $\beta$ parameter of $+\infty$ and take $C_{\alpha-\beta}(w, d-1)$ evaluations. All other $\alpha$-$\beta$ probes have a smaller $\beta$ parameter. Figure 3.5 illustrates the best case for these $\alpha$-$\beta$ probes.

The best case (i.e., the most pruning) for an $\alpha$-$\beta$ probe with $\beta = v + 1$ on a min node $\boldsymbol{a}$ occurs when the values of nodes $\boldsymbol{a} \ldots \boldsymbol{k}$ are as indicated in figure 3.5. A careful inspection of the figure indicates that in the best case, the $\beta$ parameter $v+1$ does not have influence on the pruning in the tree. The amount of pruning is therefore equal to the best case of $\alpha$-$\beta$ search with an open window on the same tree: $w^{\lfloor d/2 \rfloor} + w^{\lceil d/2 \rceil} - 1$.

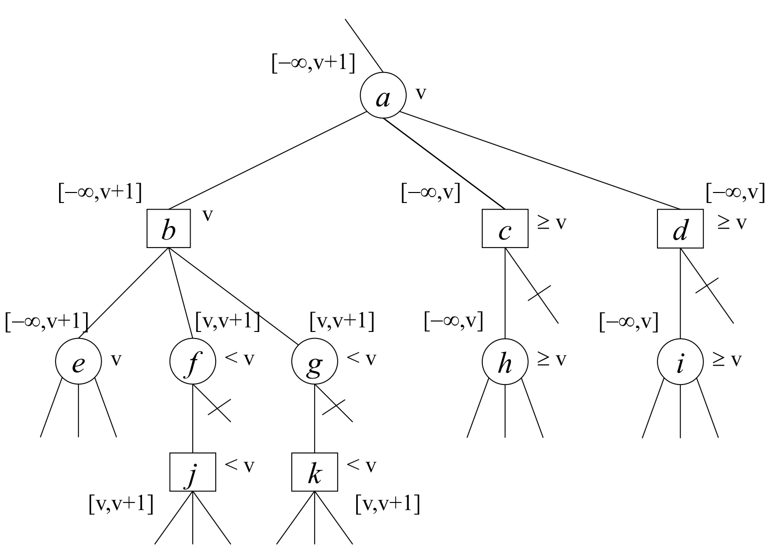


Figure 3.5: An example to illustrate the best case for $\alpha$-$\beta$ probes. The $[\alpha, \beta]$ windows and the subgame values are given next to the nodes.

As stated above, there are $w^i$ probes at every odd ply $2i - 1$ that each cost $C_{\alpha-\beta}(d - 2i + 1, w)$ evaluations. Together with the $w^{\lceil d/2 \rceil}$ evaluations for the max player, $\beta$-pruning OM search with $\alpha$-$\beta$ probes in the best case costs:

$$\begin{aligned}
C_{OM^{\beta pb}}(d,w) &= w^{\lceil d/2 \rceil} + \sum_{i=1}^{\lceil d/2 \rceil} w^i \, C_{\alpha-\beta}(d - 2i + 1, w) \\
&= w^{\lceil d/2 \rceil} + \sum_{i=1}^{\lceil d/2 \rceil} w^i \, (w^{\lfloor (d-2i+1)/2 \rfloor} + w^{\lceil (d-2i+1)/2 \rceil} - 1) \\
&= w^{\lceil d/2 \rceil} + \sum_{i=1}^{\lceil d/2 \rceil} w^i \, w^{\lfloor (d-2i+1)/2 \rfloor} + w^i \, w^{\lceil (d-2i+1)/2 \rceil} - w^i \\
&= w^{\lceil d/2 \rceil} + \lceil d/2 \rceil (w^{\lfloor (d+1)/2 \rfloor} + w^{\lceil (d+1)/2 \rceil}) - \frac{w^{\lceil d/2 \rceil + 1} - w}{w - 1}
\end{aligned} \tag{3.10}$$

### 3.3.3 A comparison

The next formulae summarize the best-case analyses for $\beta$-pruning OM search:

$$\begin{aligned}
\mathrm{OM}^{\beta 1p}&: \quad w^{\lceil d/2 \rceil} + w^{\lfloor d/2 \rfloor} + (w-1) \lceil d/2 \rceil \, w^{\lfloor d/2 \rfloor} \\
\mathrm{OM}^{\beta Pb}&: \quad w^{\lceil d/2 \rceil} + \lceil d/2 \rceil (w^{\lfloor (d+1)/2 \rfloor} + w^{\lceil (d+1)/2 \rceil}) - \frac{w^{\lceil d/2 \rceil + 1} - w}{w - 1}
\end{aligned}$$

Despite the closed forms of the functions, their relation is not immediately clear. In figure 3.6 these functions are plotted next to each other. All graphs show the value of the equation above *divided* by the best case of $\alpha$-$\beta$ search for the same depth and branching factor. Because the behaviour of the functions differs considerably for odd and even search depths, we present separate graphs for both cases (see x-axis). The two graphs on the top show the best-case complexities of OM$^{\beta 1p}$ and OM$^{\beta Pb}$ for even search depths, the two graphs at the bottom show the best-case complexities of OM$^{\beta 1p}$ and OM$^{\beta Pb}$ for odd depths.

In all cases the complexity of OM$^{\beta 1p}$ is smaller than the complexity of OM$^{\beta Pb}$. Furthermore, the complexity approximates a linear function of the (odd or even) search depth for both OM$^{\beta 1p}$ and OM$^{\beta Pb}$. It is also approximately linear in the branching factor for both OM$^{\beta 1p}$ and OM$^{\beta Pb}$, but only in the case of even search depths.

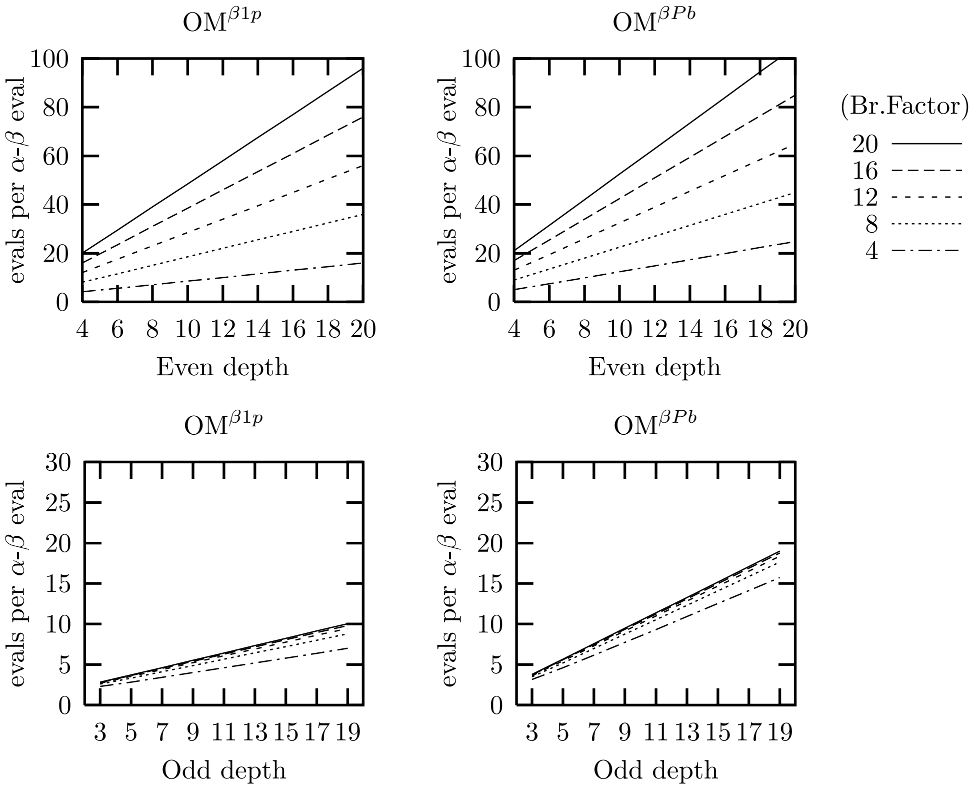


Figure 3.6: Best-case results of OM$^{\beta 1p}$ and OM$^{\beta Pb}$ compared.

## 3.4 Applicability of Search Enhancements in OM Search

The current successes of $\alpha$-$\beta$ search are largely owed to a range of search enhancements that have been added to the basic search algorithm (Hsu, 2002). In this section we will discuss the applicability of the most important search enhancements to OM search: move ordering (3.4.1), transposition tables (3.4.2), iterative deepening (3.4.3), search extensions and forward pruning (3.4.4), endgame databases (3.4.5), and aspiration search (3.4.6). We will not provide any quantifications because the application and effectiveness of a search enhancement is often dependent on a specific game and also on the presence or absence of other search enhancements. In chapter 5 some of the enhancements will be studied in more detail.

### 3.4.1 Move ordering

As discussed in section 3.3, the effect of pruning in $\alpha$-$\beta$ search depends strongly on the order in which the positions are inspected. Much effort must therefore be put on the ordering of moves during the search. In practice, move ordering is often based on game-specific heuristics, such as the rule that capture moves should be investigated before non-capture moves. To achieve the best pruning, the moves should be ordered according to the evaluation function.

For the pruning versions of OM search, the ordering of moves is also important. The best move ordering, however, diverges between the different implementations of OM search. We distinguish three cases:

(1) In $\beta$-pruning OM search with $\alpha$-$\beta$ probing, the need for ordering of moves is evident inside the probes, but this ordering must be based on the *opponent's* evaluation function. Moreover, the ordering at max nodes outside the probes is important, because the $\beta$ parameters for the probes are passed from sibling to sibling in the max nodes. This ordering should therefore be based on the opponent's evaluation function too. Also in one-pass $\beta$-pruning OM search, pruning is performed on the opponent's values, so the ordering should again be based on these values.

(2) The bounded-sum pruning algorithms are somewhat ambiguous with respect to move ordering. If $\alpha$-$\beta$ probes are used, the ordering inside those probes must be imposed by the opponent's evaluation function. Elsewhere, pruning takes place intermingled for both players, so an ordering must be found that is profitable for both evaluation functions at the same time. This task will be easier if the bound $B$ is small. If $B$ is large, pruning on MAX's evaluation function will seldom take place so ordering can concentrate on the opponent's evaluation function.

(3) In case of restricted speculation or different search depths for MAX and MIN, the move ordering on positions must depend on which evaluation function is used in the $\alpha$-$\beta$ probe: $V_0(\cdot)$ or $V_{op}(\cdot)$.

The *killer-move* heuristic (Akl and Newborn, 1977) and the *history* heuristic (Schaeffer 1983; 1989) are game-dependent move-ordering heuristics. Both heuristics are based on the observation that a good move in one position might also be a good move in another position. The killer-move heuristic stores one move per search-tree layer that was the best or caused a cut-off. The history heuristic stores the count of successes for every possible move (or move class). Both heuristics are used to reorder the moves after move generation. The applicability of both heuristics inside $\alpha$-$\beta$ probes is evident, but separate tables are needed for MAX and MIN.

### 3.4.2 Transposition tables

A *transposition table* (Greenblatt, Eastlake, and Crocker, 1967; Slate and Atkin, 1977) is a hash table in which information is stored on positions that are encountered during search. The table is indexed by a hash function that is based on the actual position, but not on history information. When a position is re-encountered during search, for instance because of a transposition in the game, the information in the transposition table can be used to prune the search tree. Transpositions are histories that lead to the same position in the game. It depends on the rules of a game whether transpositions are treated as completely identical. But even if transpositions are not totally identical, the information in the transposition table can be of use. For instance, the move that is stored in the transposition table because it was the best move, can be put in front of the move list when a transposition is encountered so that it is inspected first. It is also possible to adjust $\alpha$ and $\beta$ values on the basis of the information stored in the table.

Since the number of positions that is encountered during a search is often much larger than any transposition table can hold, a replacement mechanism (Breuker, Uiterwijk, and Van den Herik, 1994) is needed to deal with decisions whether information of old positions must be overwritten by newly encountered positions. The kind of information that is stored in the transposition table depends on the search algorithm that is used, the set of search enhancements, and on game-specific aspects. For more information on transposition tables and replacement schemes we refer to Breuker (1998). Sometimes multiple transposition tables are used for different aspects of the positions. CHESS 4.5, for instance, uses a separate transposition table for pawn structures (Slate and Atkin, 1977).

In OM search, transposition tables can help pruning the search tree in multiple ways. The most straightforward way is to incorporate transposition tables in the $\alpha$-$\beta$ probes, using the standard method as described by Breuker (1998). Since these probes are performed many times and some of them are partly repeated a number of times, it is probable that positions are reencountered during search and that information of those positions is found to be stored in the transposition table. Transposition tables therefore are likely to decrease the effort inside the $\alpha$-$\beta$ probes significantly.

In $\beta$-pruning OM search with probes, next to a transposition table used during the $\alpha$-$\beta$ probes, a separate transposition table can be used to store $v_0(h)$ values. This second table can only be used to handle real transpositions (i.e., positions that are encountered twice or more during search) since move ordering must be imposed by the opponent's evaluation function. Furthermore, no bound information on $v_0(h)$ is available, so the usual flags indicating an upper bound or lower bound are not needed in this transposition table.

An extended transposition table can be used in the one-pass versions of both $\beta$-pruning and bounded-sum pruning OM search. This transposition table could hold information on $v_0(h)$ and $v_{op}(h)$ and could include flags to indicate whether these values are lower bounds, upper bounds or exact values.

### 3.4.3 Iterative deepening

Iterative deepening is a search enhancement that allows a better move ordering with the aid of a transposition table. It also allows any-time behaviour which means that the algorithm can be terminated at any time and always gives a reasonable answer; the longer the algorithm runs, the better the answer becomes. It was first applied in the program Chess 4.5 (Slate and Atkin, 1977). The basic idea behind iterative deepening is that not a single (reduced) game $G$ is solved, but a *series* $G_1, G_2, G_3, \ldots$ of reduced games of $G$ such that $H_1 \sqsubset H_2 \sqsubset H_3 \sqsubset \ldots \sqsubset H$. The relation $H_i \sqsubset H_j$ means that $H_i$ is less deep than $H_j$: for every history $h_j \in H_j$ there is a history $h_i \in H_i$ and a sequence of moves $mm$ such that $h_j = h_i + mm$. During the searches, information on encountered positions is stored in a single transposition table, including the best move to play. In the move-ordering phase, a position is first looked up in the transposition table. If the position is found, the suggested move is put in front of the list of moves. The rationale of this is that a search with a smaller reduced game might produce good predictions of the best move to play during the next reduced game. Because the search can stop safely after every solved game $G_i$, iterative deepening makes it possible to be used in time-critical settings that request any-time behaviour.

It is clear that OM search is well-suited for iterative deepening. No major adaptations of the algorithms are needed to implement this enhancement.

### 3.4.4 Search extensions and forward pruning

The reduction of a game tree during heuristic search does not have to lead to a uniform game tree in which every history (except for histories that lead to the end of the game) has the same length. Some positions might be searched more deeply than others because they seem unstable (*quiescence search*: Shannon (1950)) or interesting (*search extensions*: Anantharaman, Campbell, and Hsu (1988)). In other positions, certain moves are not considered because there are too many moves (*selective search*: Shannon (1950), Wilkins (1982), Kaindl, Horacek, and Wagner (1986)), some of the moves are clearly too bad to be played (*forward pruning*: Shannon (1950), Smith and Nau (1994)), or a shallow search starting with a pass indicates that expansion is not needed (*null move*: Beal (1989)). The application of these enhancements requires game-specific knowledge.

It is possible to apply these techniques in OM search, but care should be taken that the opponent's strategy still can be predicted as good as possible. It may happen that the opponent is actually using different search extensions or a different forward-pruning technique than max is assuming. Algorithm 3.5 covers the differences between max and min and is in principle suitable for the search enhancements discussed above.

### 3.4.5 Endgame databases

Endgame databases (Ströhlein, 1970; Van den Herik and Herschberg, 1985) store pre-computed game-theoretic values of positions near the end of a game. The tables can store, for instance, in how many moves a game can be won (*distance to win*) or in how many moves a piece can be captured or promoted (*distance to conversion*). Since most endgame databases contain information about board positions, not about histories, it is not trivial to convert the endgame-database value of a position into the game-theoretic value of a history. Nevertheless, endgame databases can be seen as part of the evaluation function. The availability of endgame databases for the opponent is therefore part of the opponent modelling in OM search.

### 3.4.6 Aspiration search

Aspiration search (Brudno, 1963; Pearl, 1980) is an application of $\alpha$-$\beta$ search in which the bounds $\alpha$ and $\beta$ at the root of the search tree are not set to the 'open window', $[-\infty, +\infty]$, but to a narrower window. The smallest window applied is the 'null window' (Baudet, 1978): $[\alpha, \alpha+1]$. When a call to $\alpha$-$\beta$ search at the root level returns with a value within the interval $[\alpha, \beta)$ then this is the true value of the root. Otherwise a re-search is needed with another window. Search with a small window is fast since many branches of the game tree will be pruned. Two popular applications of aspiration search (with null windows) that also involve transposition tables and iterative deepening are: PV search (Marsland, 1983) and MTD(f) search (Plaat, 1996). PV search (principal variation search) assumes that the first move at every position is the best move to play. After the evaluation of the first move, the other moves are searched with a null window at the current $\alpha$. If the return value is lower than or equal to $\alpha$, the move is disregarded; else the move is re-searched with a larger window. MTD(f) search (Memory-enhanced Test Driver) applies a series of null-window searches at the root of the search tree in order to find the true value, using binary search. In combination with iterative deepening and transposition tables, MTD(f) search is equivalent to a best-first search method.

The $\beta$-pruning variants of OM search only have a $\beta$ parameter, so there is no role for null windows. The bounded-sum versions, however, could be enhanced with some form of aspiration search at the root level, but since the value for MAX and MIN are expected to be different, two separate windows have to be used. Furthermore, the effect of the aspiration search will depend on the bound $B$ because this bound 'opens up' the window which causes less pruning.

When $\alpha$-$\beta$ probes are used, it is theoretically possible to apply MTD(f) search to the probes, but the binary search has to be performed at every probe separately. It is likely that the extra overhead will outweigh the advantages of MTD(f) search. The application of PV search is appropriate inside the $\alpha$-$\beta$ probes. The negative effect of multiple execution of parts of the probes is diminished when PV search is used because the information in the transposition tables of a previous probe will lead to a good move ordering and less re-searches. When PV search is applied in the $\alpha$-$\beta$ probes, it is possible to add, for instance, an enhancement called *multi-cut* pruning (Björnsson, 2002) that allows further forward pruning. This enhancement decides on the basis of a series of shallow pre-searches to prune a node if more than a given number of moves in that node are likely to lead to a violation of the current $\alpha$ or $\beta$ boundary.

## 3.5 Chapter Conclusions

In this chapter we introduced two different approaches to OM search: a one-pass approach and an approach that involves $\alpha$-$\beta$ probes. Below we summarize the conclusions arrived in this chapter. We showed that like in $\alpha$-$\beta$ search, pruning is also possible in OM search. However, the amount of pruning is limited because in general pruning is only possible at max nodes ($\beta$ pruning). If the difference between evaluation functions is bounded, pruning is possible at min nodes, too (bounded-sum pruning). The best-case study of $\beta$-pruning OM search showed that the one-pass approach is optimal in the best case. Both approaches (one-pass and $\alpha$-$\beta$ probing) show a (nearly) linear relation with $\alpha$-$\beta$ search with respect to the number of necessary evaluations. Finally, we studied a range of well-known search enhancements of which most appeared to be applicable to OM search, especially within the framework of $\alpha$-$\beta$ probes. The enhancements are predominantly guided by the opponent's evaluation function.

# Chapter 4

# Opponent-Model Search: Evaluation and Admissibility

*The previous chapter dealt with the efficient implementation of OM search. In this chapter we concentrate on the effectiveness of the search method. Obviously, the evaluation functions play a major role. At first glance, OM search should only be applied if the own evaluation function is better than the opponent's one. Since the notion 'better' is unclear, we analyse a variety of relations between evaluation functions more closely. A further analysis of OM search reveals a rather unexpected condition on the pair of evaluation functions that is important for the effectiveness of OM search: admissibility.*

---

**Chapter contents:** This chapter concentrates on the evaluation functions that are used in OM search. Section 4.1 is a study of possible relations between evaluation functions. Section 4.2 describes to a restriction on pairs of evaluation functions, called admissibility, that can increase the effect of OM search. Finally, section 4.3 provides the chapter conclusions.[1]

## 4.1 Evaluation Functions in OM search

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 one, since in such a case it seems better to adopt the opponent's evaluation function and use Minimax search. However, the term 'better' is not easy to define.

[1]The contents of this chapter have been published in Donkers, H.H.L.M., Uiterwijk, J.W.H.M., Herik, H.J. van den (2003). Admissibility in Opponent-Model Search. *Information Sciences*, Vol. 154, Nos. 3–4, pp. 119–140.

In this section we will look in detail to evaluation functions and, in particular, how evaluation functions can be compared to each other.

### 4.1.1 Analysis of evaluation functions

In chapter 1 we introduced the concept of a static heuristic evaluation function. Static heuristic evaluation functions (or evaluation function 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 reduced game). For an introduction to heuristics in general and to static evaluation functions in particular, we refer to Pearl (1984).

An evaluation function returns a scalar value that measures the strength of that position for MAX. If a position happens to be an end 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 proper draw is rewarded by 0; immediate win for MAX is rewarded by some large value $W > H$; and immediate 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 heuristic values can be interpreted in three different ways. The first interpretation is to view the heuristic value as a *predictor* for the true game-theoretic value: the higher the heuristic value for a position, the higher the chance that the position is a game-theoretic win. This interpretation is used by Pearl (1984) and others in their theoretical study of Minimax search and $\alpha$-$\beta$ search.

The second interpretation is to view the heuristic value as some encoding of the *probability* to win the game at that position. This interpretation is used by authors that use machine-learning techniques like temporal-difference learning to achieve evaluation functions (cf. Baxter, Trigdell, and Weaver, 1998). The probability to win the game from a given position, however, depends on the specific opponent and on a number of game settings like the start position, the search time, and the search technique used.

The third interpretation is to view the heuristic value as a measure of the *profitability* of the position for MAX. The profitability of a position is only partly based on the prediction of the game-theoretic value. It also incorporates the possibility of errors on both sides, independent of a specific opponent. In addition, it includes the own strengths and weaknesses of MAX. The profitability of a position is independent of the actual opponent and of the actual game setting. For example, a position that is a game-theoretic win, but from which the winning solution is difficult to reach because of many threats and many possible errors, is not profitable. In contrast, a position that is a game-theoretically loss could offer so many opportunities to exploit an opponent's error, that the position may become profitable. This third interpretation is, in our view, used in hand-made, knowledge-based evaluation functions, although during tournaments, evaluation functions are often fine-tuned to specific circumstances or to a specific opponent. In the sequel, we will use the profitability interpretation of evaluation functions, unless stated otherwise.

It is a complication that the profitability of a position is just as difficult to determine as the game-theoretic value itself. An evaluation function can only *estimate* the profitability of a position. Furthermore, the profitability of a position is based on a whole set of scalar and symbolic properties of a position. All these properties have to be projected together onto a single scalar value.

### 4.1.2 Comparing 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 appear to be possible: below we will define eight different orderings.

The first way in which evaluations can be ordered is on the degree in which they predict the game-theoretic values. Assume that the true payoffs in a game $G$ are restricted to: $\{win, loss, draw\}$ and that the ranges of the heuristic evaluation functions are restricted to the interval $[-W, W] \subset \mathbb{Z}$.[2] Let $\mathbf{V} : E' \to \{win, loss, draw\}$ denote the true game-theoretic value of the end positions in the reduced game $G'$ and let $\mathcal{V}$ denote the set of all evaluation functions $V : E' \to [-W, W]$.

**Definition 1.** An evaluation function $V$ is called a *perfect predictor* if a function $t : [-W, W] \to \{win, loss, draw\}$ exists such that for all $h \in E'$ holds that $t(V(h)) = \mathbf{V}(h)$.

A perfect predictor makes any search superfluous, provided that $t$ is known.

**Definition 2.** An evaluation function $V$ is called a *partial predictor* if function $t$ and a subset $Q \subset E'$ exist such that for all $h \in Q$ holds that $t(V(h)) = \mathbf{V}(h)$. Any such subset $Q$ is called a *perfectly predicted set for $V$*.

The definition of a perfect predictor is equivalent to the definition of *perfect play* by Christensen and Korf (1986). In their definition, an evaluation function performs perfect play if it exhibits 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.

The first ordering of evaluation functions is based on the size of the set on which an evaluation function is a perfect predictor.

**Ordering 1.** Evaluation function $V_1 \in \mathcal{V}$ is called a *larger predictor* than function $V_2 \in \mathcal{V}$ if $V_1$ has a larger perfectly predicted set than $V_2$.

It is possible to define an ordering of evaluation function on the basis of the following domination relation, but since this is only a *partial* ordering[3], it will not be of major use.

[2]We disregard real-valued evaluation functions since $E'$ is a finite set. Any evaluation function $f : E' \to \mathbb{R}$ can be mapped isomorphically to a function $V : E' \to \mathbb{Z}$ such that $f(h_1) \leq f(h_2) \Leftrightarrow V(h_1) \leq V(h_2)$.

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

**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 the perfectly predicted set of $V_2$ is a proper subset of the perfectly predicted set of $V_1$.

In the case of most games that are of interest, the 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 more usefully, the mapping function $t$ must be relaxed. One way to relax $t$ is to add some probability and statistics: take a sample $E_i \subseteq E'$ of end 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 difference between win positions and loss positions and use this P-value to order the evaluation functions. This leads to the third 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 $V_1$ provides a more significant prediction of the game-theoretic value than $V_2$.

The three orderings above concentrated on the predictability of the evaluation function for the game-theoretic value. However, as already noted, a prediction of the game-theoretic value might be of less use than a prediction of the profitability, unless the prediction is perfect.

The fourth way in which evaluation functions can be ordered is based on their operational behaviour.

**Ordering 4.** Evaluation function $V_1$ is called *operationally better* than evaluation function $V_2$ if a player $A$ that uses $V_1$, receives a higher score in a given tournament than a player $B$ that uses $V_2$ instead of $V_1$, but is equal to $A$ in all other aspects.

The set-up of the tournament influences the exact meaning of this operational relation between evaluation functions. Important factors are the set of used opening positions, the search depth, and the size of the tournament: are other players also included or are these two players only playing against each other? The operational relation between evaluation functions is an overall measurement of the estimation that the evaluation functions give for the profitability of the positions. The operational relation is one of the orderings used in the field of automatically learning evaluation functions (for instance, in genetic algorithms and temporal-difference learning). Another ordering often used in evaluation-function learning, is related to *move invariance* as explained by Christensen and Korf (1986). 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 (1959; 1967) used this ordering implicitly in his pioneering work on the game of checkers.

The relation between evaluation functions that is of most use in the discussion on OM search below is based on their estimation of the profitability. This relation is quite opposite to the operational one since profitability is not measurable. For a start, we have to assume that an evaluation function $\mathbb{V} : E' \rightarrow \mathbb{Z}$ exists that measures the real *profitability* of all end positions $h \in E'$. The higher the value of $\mathbb{V}(h)$, the higher the profitability is. First we give a general definition.

**Definition 3.** Two evaluation functions $V_1$ and $V_2$ are *equivalent* on $E$ if they order the positions equally: $\forall h_a, h_b \in E' : V_1(h_a) \leq V_1(h_b) \Longleftrightarrow V_2(h_a) \leq V_2(h_b)$.

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

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

Just like the perfectly predicting evaluation function, a perfect evaluation function in this sense removes the need for any search. Unfortunately, one never knows when an evaluation function is perfect. 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 $\mathbb{V}$.

**Ordering 5.** Evaluation function $V_1$ is called a *larger profitability estimator* than evaluation function $V_2$ if it equivalent with $\mathbb{V}$ on a larger subset of $E'$ than $V_2$.

Also in the case of profitability estimation, a partial ordering of evaluation functions can be defined on the basis of a domination relation.

**Ordering 6.** Evaluation function $V_1$ is a *dominating profitability estimator* with respect to $V_2$ if it is at least equivalent with $\mathbb{V}$ for all positions for which $V_2$ is equivalent with $\mathbb{V}$, but also is equivalent with $\mathbb{V}$ for other positions.

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

**Ordering 7.** Evaluation function $V_1$ is called a *statistically better profitability estimator* than evaluation function $V_2$ simply 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*[4] 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 $\sum_h |rank(h, V) - rank(h, \mathbb{V})|$. This produces the eighth ordering of evaluation functions.

**Ordering 8.** Evaluation function $V_1$ is called a *better ranked profitability estimator* than evaluation function $V_2$ 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

[4]This is similar to Spearman's Rank Correlation Coefficient (Spearman, 1904), which also can be used here.

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 $V$ *overestimates* a position $h$ if the rank of $V(h)$ is higher than the rank of $\mathbb{V}(h)$.

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

The analysis of evaluation functions and their ordering makes it clear that a simple use of a relation 'better' with respect to evaluation functions is not possible. Which one of the following orderings should be used?

(1) $V_1$ is a larger 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 larger 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). However, the orderings are not equivalent. The selection of one of the orderings should be made by a three-step procedure: (1) the manner in which the evaluation functions are constructed, (2) the purpose of the comparison of evaluation functions, and (3) the practical circumstances.

## 4.2 Estimation errors in OM search

In this section[5] we study the effect of estimation errors (underestimations and overestimations) in both evaluation functions $V_0$ and $V_{op}$ on the outcome of OM search. This will lead to a condition on the pair of evaluation functions, called admissibility, that is related to the rank-based ordering (8) in the previous section.

### 4.2.1 An alternative view on OM search

In OM search, underestimations and overestimations can have a large influence on the outcome of the search. In order to illustrate this, we use an alternative view on OM search. Let $G$ be the game tree under consideration by OM search. At every min node, MIN determines which branch is selected. Let $G_{min}$ be the subtree of $G$ in

[5]This section has been published in Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2003). Admissibility in Opponent-Model Search. *Information Sciences*, Vol. 154, Nos. 3–4, pp. 95–202.

which all branches at min nodes are removed except for the branches selected by MIN (i.e., the solution tree for MIN), and let $E_{min}$ be the set of positions corresponding with the leaves of $G_{min}$. The remaining task of MAX is to find the position $h^* \in E_{min}$ that has the largest value of $V_0$.

Obviously, evaluation function $V_{op}$ determines the set $E_{min}$, and $V_0$ determines $h^*$. Assume that $V_0$ and $V_{op}$ start as perfect evaluation functions ($V_0 = V_{op} = \mathbb{V}$). Assume further that we introduce *estimation errors* in $V_0$ and/or $V_{op}$ in the sense of the definitions 5 and 6 in section 4.1.2. These are modifications in the evaluation functions that change the rank of the positions. We define estimation errors in both $V_0$ and $V_{op}$ from the perspective of MAX. So, when the rank of $V_{op}(h)$ is lower than the rank of $\mathbb{V}(h)$, then this is called an *underestimation* although $V_{op}$ is judging the position too beneficial for MIN. Likewise, when the rank of $V_{op}(h)$ is higher than the rank of $\mathbb{V}(h)$, then this is called an *overestimation* although $V_{op}$ is judging the position too harmful for MIN.

Estimation errors in $V_0$ and $V_{op}$ only have effect on the outcome of OM search if they influence $E_{min}$ or $h^*$. There are four types of estimation errors that can lead to such an effect. They are:

| | | |
|---|---|---|
| Type-I error | : | $V_0$ overestimates a position in $E_{min}$ |
| Type-II error | : | $V_0$ underestimates a position in $E_{min}$ |
| Type-III error | : | $V_{op}$ underestimates a position that thus enters $E_{min}$ |
| Type-IV error | : | $V_{op}$ overestimates a position in $E_{min}$ |

## Type-I error

A type-I error is characterized by the fact that $V_0$ overestimates a position in $E_{min}$. It is the most serious error of OM search. $V_0$ can overestimate any of the positions

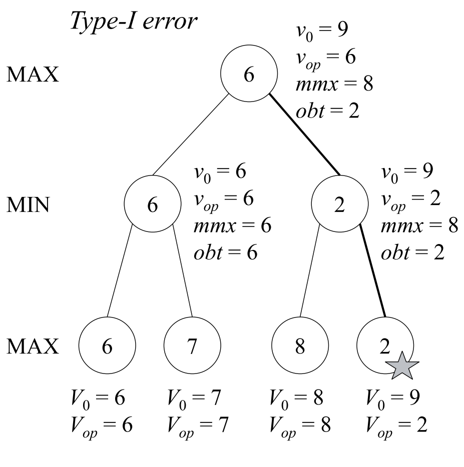

Type-I error
MAX
MIN
MAX
$v_0 = 9$, $v_{op} = 6$, $mmx = 8$, $obt = 2$
$v_0 = 6$, $v_{op} = 6$, $mmx = 6$, $obt = 6$
$v_0 = 9$, $v_{op} = 2$, $mmx = 8$, $obt = 2$
6
6
2
6
7
8
2
$V_0 = 6$, $V_{op} = 6$
$V_0 = 7$, $V_{op} = 7$
$V_0 = 8$, $V_{op} = 8$
$V_0 = 9$, $V_{op} = 2$

Figure 4.1: Example of a type-I error.

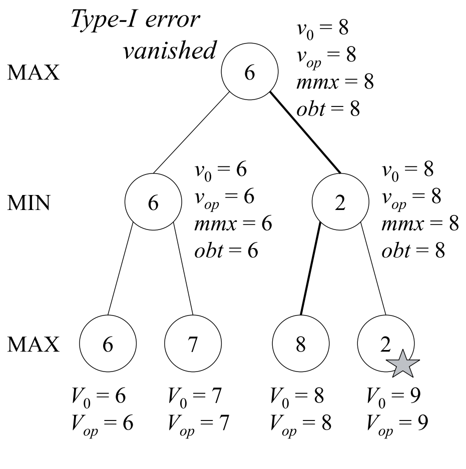

Type-I error vanished
MAX
MIN
MAX
$v_0 = 8$, $v_{op} = 8$, $mmx = 8$, $obt = 8$
$v_0 = 6$, $v_{op} = 6$, $mmx = 6$, $obt = 6$
$v_0 = 8$, $v_{op} = 8$, $mmx = 8$, $obt = 8$
6
6
2
6
7
8
2
$V_0 = 6$, $V_{op} = 6$
$V_0 = 7$, $V_{op} = 7$
$V_0 = 8$, $V_{op} = 8$
$V_0 = 9$, $V_{op} = 9$

Figure 4.2: Example of a type-I error vanishing.

in $E_{min}$ and subsequently select it as a maximum. A type-I error vanishes if $V_{op}$, too, overestimates the same position (already overestimated by $V_0$) and does so by a sufficiently large margin. The overestimation of the position by $V_{op}$ causes it to be removed from $E_{min}$.

Figure 4.1 gives an example of a type-I error. MAX is in error, but believes MIN is in error. In the figure, the numbers inside the nodes give the true values (the $\mathbb{V}$ values) of the profitability of the nodes. Next to the nodes the values of $V_0$ and $V_{op}$ (or $v_0$ and $v_{op}$ for the internal nodes) are given. Two additional values are presented at an internal node: *mmx* is the minimax value that MAX would obtain if $V_0$ was used with Minimax search, and *obt* gives the truly obtained profitability for MAX, e.g., the true value of the chosen variation. In the example a type-I error is generated by MAX's overestimation of the right-most leaf. This causes that MAX would *expect* to obtain a value of 9 but in reality, MAX only obtains 2, which is 4 points fewer than the true profitability of the root.

The tree in figure 4.2 shows how this error might vanish. If MIN sufficiently overestimates the same position that MAX overestimates, the position will not be selected by MIN anymore. The difference between believed and obtained value disappears (both values are now 8). Type-I errors are severe since the overestimation acts as an *attractor*: the larger the error in $V_0$, the larger the probability that it changes the value of $v_0$ at the root of the tree.

## Type-II error

A type-II error is characterized by the fact that $V_0$ underestimates a position in $E_{min}$. It is a less serious error for MAX than a type-I error because it only results in another move if the best position $h^*$ is underestimated (figure 4.3). Furthermore, if $h^*$ is strongly underestimated, the second-best position will take its place and

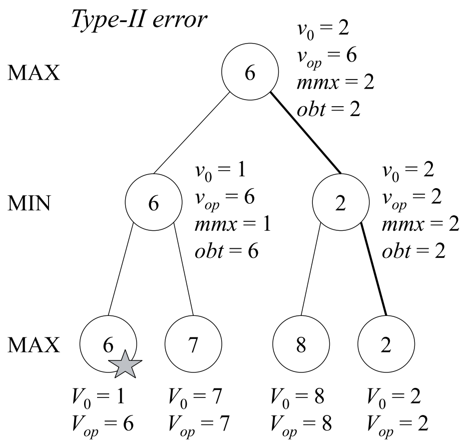


Figure 4.3: Example of a type-II error.

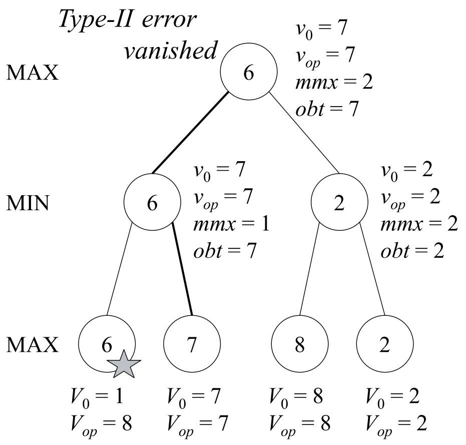


Figure 4.4: Example of a type-II error vanishing.

the overall harm is limited. Type-II errors vanish when $V_{op}$ overestimates the same positions, causing these positions to disappear from $E_{min}$. See figure 4.4, where MIN evaluates the left most leaf at 8 instead of at 6.

### Type-III error

A type-III error is characterized by the fact that $V_{op}$ underestimates a position that thus enters $E_{min}$. It is the major error for MIN. By underestimation, any position could enter $E_{min}$, even positions that are unprofitable for MIN (see figure 4.5). MAX can profit the most from this error. The error vanishes if $V_0$ underestimates these positions too (figure 4.6).

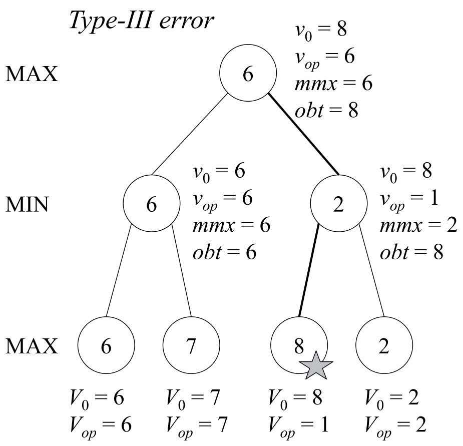


Figure 4.5: Example of a type-III error.

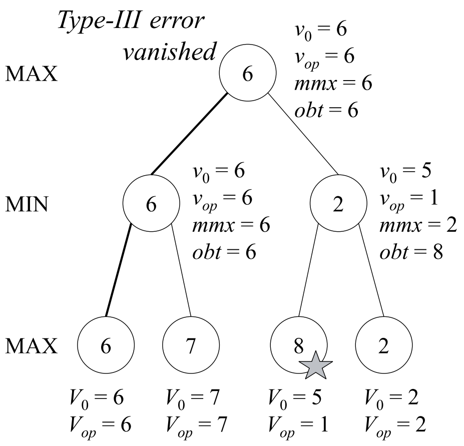


Figure 4.6: Example of a type-III error vanishing.

### Type-IV error

Finally, a type-IV error is characterized by the fact that $V_{op}$ overestimates a position in $E_{min}$. It is less serious for MIN than a type-III error. It may cause a position to disappear from $E_{min}$ (see figure 4.7). MIN is not able anymore to profit from opportunities in this position, but a second-best position will take its place. This type cannot vanish by a change in $V_0$.

### The effect of errors

Errors due to overestimation and underestimation are not unique to OM search, but the effect of these errors in Minimax is bounded by the value of siblings, just like the errors of type II and IV above. In Minimax search, a single underestimation or overestimation will, in general, not do much harm, whatever the size of the error (see, for instance, Pearl (1984)). The example in figure 4.1 illustrates this observation. The overestimation of the right-most leaf with any amount larger than

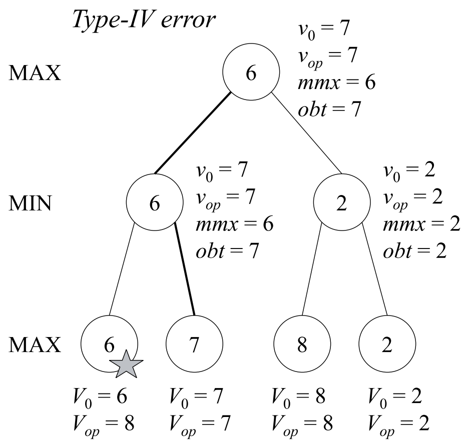


Figure 4.7: Example of a type-IV error.

8 causes Minimax to select the sibling of this node. Once this sibling is selected, the extent of the overestimation does not matter anymore. However, this Minimax-search behaviour is in clear contrast with the OM-search behaviour for errors of type I. There, the extent matters seriously. The value $v_0$ of the root depends on this overestimated value. If the example tree is a subtree of a larger search tree, the root value $v_0$ will be propagated upwards within the larger search tree. The larger the overestimation, the larger the damage will be in the search tree. A single large type-I error can thus cause major damage to the position of MAX. (In Minimax search, the larger the overestimation, the more likely it is that it will not be selected at one of the parent min nodes. Therefore it will not easily propagate upwards in the tree.) Fortunately, there is also another side of this coin: a single large type-III error can lead MAX to a considerable win.

### 4.2.2 Admissibility

For OM search to be at least as successful as Minimax, type-I errors are undesirable. In fact, the relation between $V_0$ and $V_{op}$ must be such that type-I errors do not (or are rather unlikely to) occur. Furthermore, type-III and type-IV errors should occur as often as possible in order to let OM search profit from them. This leads to the following *admissible-pair* conditions on the evaluation functions $V_0$ and $V_{op}$.

**Definition 7.** A pair of evaluation functions $(V_0, V_{op})$ is *admissible for OM search* if (1) $V_0$ is a better profitability estimator than $V_{op}$, and (2) $V_0$ never overestimates a position that $V_{op}$ does not overestimate likewise.

The condition in definition 7 seems disputable, because it aims to avoid only type-I errors. However, these errors are the most severe ones. Furthermore, the condition is not sufficient to prevent type-I errors completely, but under the current conditions they can occur only when the quality of $V_0$ is low. In this case the prevention of type-I errors might fail because it can happen that all children of one min node are seriously overestimated, and one of them enters $E_{min}$. However, in such case, Minimax search would yield bad results too.

The notion of admissibility in OM search is analogous to the notion of admissibility in the heuristic single-agent search algorithm $A^*$ (cf. Pearl, 1984). In order for $A^*$ to lead to an optimal solution, the heuristic function $h()$ must be *admissible*, which means that $h()$ should never *overestimate* the real distance ($h^*()$) to the goal. This condition on $h()$ is quite similar to the conditions stated above. Pearl (1984) provides a suggestion on how to discover an admissible function $h()$: create a simplified model of the problem domain by either relaxing or over-constraining the original model of the problem domain. Then use the distance function of the simplified model as an estimator of the true distance. The construction of the simplified model must be such that it can be proven that $h()$ is admissible.

A similar approach is also possible in relation to OM search. When an evaluation function $V_0$ is based on some analytical model of profitability, an opponent's evaluation function $V_{op}$ can be constructed by a careful manipulation of this analytical model. Admissibility can be reached when in the opponent model some parts of the original model are disregarded in such a way that it leads to additional overestimations, but no existing overestimations are removed. An example would be to remove the capability from the evaluation function to recognize some kind of threat.

### 4.2.3 Admissibility and bounded-sum pruning OM search

When the pair of evaluation functions $V_0$ and $V_{op}$ obey the bounded-sum property (i.e., $\forall h \in H : |V_0(h) - V_{op}(h)| \leq B$), and the bound $B$ is small, the danger of type-I errors is likely to be of minor importance, since a small difference in the value of the evaluation functions will mostly coincide with a small difference in the rank of positions. Consequently, the error will have a lower probability to propagate to the root of a larger search tree. However, the advantages of type-III errors for MAX are also limited for the same reason. So, for pairs of evaluation functions with a bounded sum, the absence of admissibility is not very dangerous when OM search is used, but the extent to which MAX can profit from MIN's errors is probably limited too.

In order to increase pruning it is possible to transform a pair of evaluation functions (without changing the order of moves) so that their sum bound $B$ is diminished. However, such an operation would not decrease the effect of estimation errors, since the transformation does not influence the rank of the moves and therefore does not change the estimation errors themselves.

## 4.3 Chapter Conclusions

In this chapter we concentrated on the role of evaluation functions in OM search. We started with a study of different manners in which evaluation functions can be compared. In the process we developed the notion of profitability. Although the profitability of a position is not explicitly measurable, this notion plays a central role in the theoretical discussion of estimation errors in OM search. We defined four types of errors in OM search and concluded that type-I errors are the most dangerous for MAX and that type-III errors are the most advantageous for MAX. The analysis of these errors leads to the definition of admissibility as a relation between evaluation functions that can prevent type-I errors.

The last two chapters gave only a theoretical analysis of OM search. The next chapter will be dedicated to the empirical study. The experiments must show what the efficiency of OM search under real-world conditions is and whether usage of an opponent model by OM search really leads to a better performance.

# Chapter 5

# Experiments in OM Search

*The proof of the pudding is in the eating. After the theoretical analysis of OM search in chapters 3 and 4, we will test the performance of the search method in some actual games in this chapter. We start with experiments on random games trees to gain insight into the average case for the computational efficiency. Experiments in the game of lines of action show what can happen if admissibility is not respected. Experiments in a chess endgame reveal how OM search may perform when admissibility is guaranteed. Finally, we use experiments in the game of bao to investigate under which circumstances OM search becomes successful.*

---

**Chapter contents:** This chapter describes four series of experiments with OM search. Section 5.2 is dedicated to random game trees, section 5.3 to experiments in lines of action, section 5.4 to experiments in a chess endgame, KQKR, and section 5.5 to experiments in the game of bao. The chapter ends with conclusions in section 5.6.[1]

## 5.1 Four Different Domains

This chapter is an empirical follow-up to the first four theoretical chapters. We performed four series of experiments, in four different domains (random game trees, lines of action, chess, and bao), each concentrating on a different aspect of OM search. Below we provide an overview of their relations.

[1]Parts of this chapter have been published in Donkers, H.H.L.M., Uiterwijk, J.W.H.M., Herik, H.J. van den (2001), Probabilistic Opponent-Model Search, *Information Sciences*, Vol. 135, Nos. 3–4, pp. 123–149, in Donkers, H.H.L.M., Uiterwijk, J.W.H.M., Herik, H.J. van den (2003), Admissibility in Opponent-Model Search, *Information Sciences*, Vol. 154, Nos. 3–4, pp. 119–140, and in Donkers, H.H.L.M., Herik, H.J. van den, and Uiterwijk, J.W.H.M., (2003), Opponent-Model Search in Bao, *Advances in Computer Games 10* (eds. H.J. van den Herik, H. Iida, and E. Heinz), Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 307–323.

In chapter 3 we introduced two versions of $\beta$-pruning OM search. We analysed the computational complexity of these algorithms for the best-case game trees. In section 5.2 we will analyse the computational complexity of $\beta$-pruning OM search on random game trees in order to gain insight into its average-case complexity.

The theoretical analysis in chapter 4 suggested that the notion of admissible pairs of evaluation functions is important in OM search. In section 5.3, experiments in the domain of lines of action are described that indicate what can happen when OM search is applied without taking into account the conditions of admissibility, even if perfect knowledge of the opponent's strategy is available. The results of these experiments have led us to the current definition of admissibility. The chess experiments in section 5.4 were conducted in a setting in which admissibility was provided by an endgame database. The results for OM search are not convincing, which means that there must be other aspects than admissibility alone that influence the effectiveness of OM search. A thorough investigation is provided.

Finally, the bao experiments in section 5.5 were used to study the circumstances in which OM search can become successful, that is, circumstances in which a player that uses OM search wins more games than a player that uses Minimax. In the concluding section 5.6 we end this empirical chapter with a listing of the circumstances under which OM-search can be used successfully.

## 5.2 Experiments with Random Game Trees

The first series of experiments with OM search were intended to give insight into the average-case computational complexity of the two basic implementations of the search method: $\mathrm{OM}^{\beta 1p}$ and $\mathrm{OM}^{\beta Pb}$. In order to obtain a game-independent measurement, we used randomly generated game trees. These random game trees are only a rough approximation of real games. Because of the absence of move ordering in these trees, the computational complexity measured in the experiments is probably larger than in real game trees.

An important aspect of average-case analysis is the reliable generation of random numbers. Our random game-tree experiments have been performed using the programming language Java. The standard random-number generator in Java uses linear recurrence with a period of $2^{48}$ (Sun, 2002). However, we apply an algorithm with much better properties: the *Mersenne Twister* (Matsumoto and Nishimura, 1998), also called MT19937, made available in a Java implementation by Luke (2002). This random-number generator has a period of $2^{19937}$. The large period makes the Mersenne Twister suitable for the generation of many random game trees with a large number of nodes. For an overview and test of random-number generators including the Mersenne Twister we refer, for instance, to L' Ecuyer (2001) and Johnson and Lancaster (2000).

In subsection 5.2.1 we will first discuss how the random game trees are constructed. Then in subsection 5.2.2 we will show how the number of evaluations is distributed over a population of random game trees. In subsection 5.2.3 the effect of search depth and branching factor on the complexity is studied and in subsection 5.2.4 the effect of transposition tables. In subsection 5.2.5 we close the section with a discussion of the results.

### 5.2.1 Construction of random game trees

In the experiments we applied OM search on random game trees. All end positions of a tree had the same depth and all internal nodes had the same number of children (branching factor or width). With every experiment the tree depth and the branching factor is specified.

The evaluation function in our game trees was generated analogously to the approach by Newborn (1977). In his approach, every node in the tree received a random number. The value of a leaf node was the average of all random numbers at the nodes on the path from the leaf node to the root of the tree. This assured some correlation between the values of nodes in neighbouring parts of the tree, partly simulating real game trees.

We adapted Newborn's (1977) approach slightly because for experiments with OM search, we needed two evaluation values per leaf node. Therefore our approach assigned two different random numbers (called base values) to every node in the tree: $base_0$ and $base_{op}$. These base values were drawn uniformly from [0,1]. Instead of taking the average of all random numbers on a path, our approach used the sum of these numbers. For $V_0$ it used the sum of the $base_0$ values, and for $V_{op}$ the sum of the $base_{op}$ values. The base values in the nodes were of float precision, the evaluation values were integer values obtained by multiplying the sum by 1000 and then rounding off the result to the nearest integer.

In order to allow paired comparisons of the search methods, we wanted to apply the different methods to exactly the same game tree. The dimensions of the trees in our experiments ranged to depth 16 and width 20, so the number of nodes per tree could become large, which made it impossible to keep these game trees in memory. Furthermore, since we wanted to test the efficiency of pruning on these random game trees, we expected that large parts of these trees would not be visited. Altogether this meant that we needed an algorithm that generated the random game tree during search and assigned random numbers to the nodes independently of pruning. Moreover, we assumed that all search methods used the same ordering of the nodes.

Figure 5.1 illustrates our approach. The generation of a random tree starts with an external random seed. Every time that the same external seed is used, exactly the same random game tree will be produced. We use the value of $base_0$ of a node as random seed for the generation of random numbers for all its direct child nodes. The generation of these numbers takes place at the move-generating step in the algorithm (e.g., line 2 in algorithm 3.2: $L \leftarrow M(h)$).

There is a potential problem with our approach of using the value of $base_0$ as random seed for the child nodes. Namely, at the same time the random-number generator uses its internal state that generated $base_0$ to produce $base_{op}$. It is therefore possible that there exists a measurable dependency between the value of $base_{op}$ and the value $base_0$ of the first child, and between subsequent pairs of generated random numbers.

In order to check whether such dependency exists in case of the Mersenne Twister, we compared the distribution of the (Minimax) root values for two sets of 10,000 random game trees each. One set of trees was generated with the above approach,

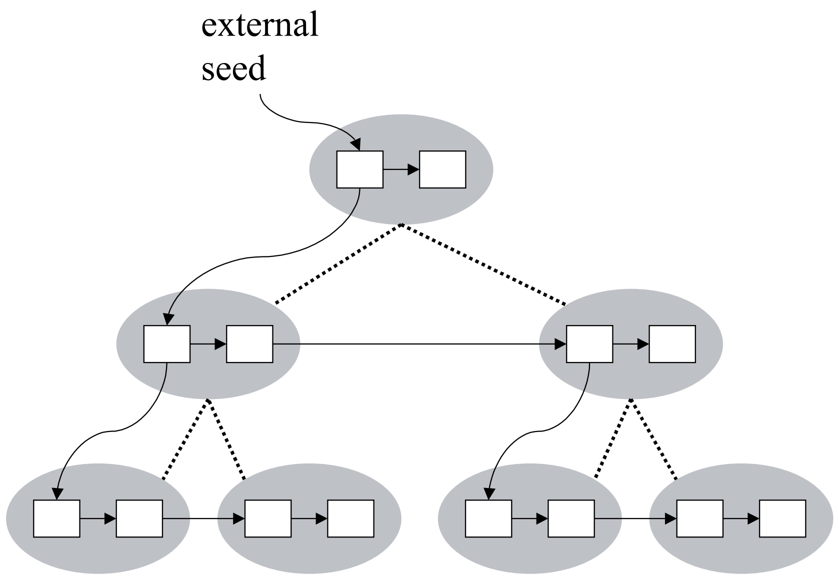


Figure 5.1: Flow of random-number generation in a random game tree. The straight arrows indicate the order of 'nextfloat' commands, the curved arrows indicate an additional 'setseed' command. The squares inside the nodes are the base numbers for the two evaluation functions.

and one set of trees was generated with a method that did not use the 'setseed' commands but generated random numbers in one stream for all nodes in the game tree. The dimensions of the trees were: depth 6 and width 3. We applied a Student-t test for unequal variations on the results. The t-value was 0.34 at a critical value of 1.96, so we could not reject the hypothesis (at a confidence level of 95%) that both samples were taken from the same distribution. This result ensured us that it was safe to use our approach.

### 5.2.2 Distribution of evaluation counts

The first experiment that we conducted on random game trees concerned the distribution of the number of nodes that is evaluated by the one-pass version and the probing version of $\beta$-pruning OM search compared to $\alpha$-$\beta$ search. Figure 5.2 shows the results for 10,000 trees of depth 8 and width 4. The distributions are roughly shaped as a normal distribution, but the tails reveal that they are slightly skewed to the right. In this setting, OM$^{\beta Pb}$ used about 3 times more evaluations than $\alpha$-$\beta$ search, and OM$^{\beta 1p}$ needed more than 4 times the amount that $\alpha$-$\beta$ search used, as can be deduced from the centres of the three distributions. The question is whether these ratios hold for all individual random game trees.

Since we applied $\alpha$-$\beta$ search, OM$^{\beta 1p}$, and OM$^{\beta Pb}$ all to the same 10,000 trees, it is possible to show the *relative* number of evaluations that OM$^{\beta 1p}$ and OM$^{\beta Pb}$

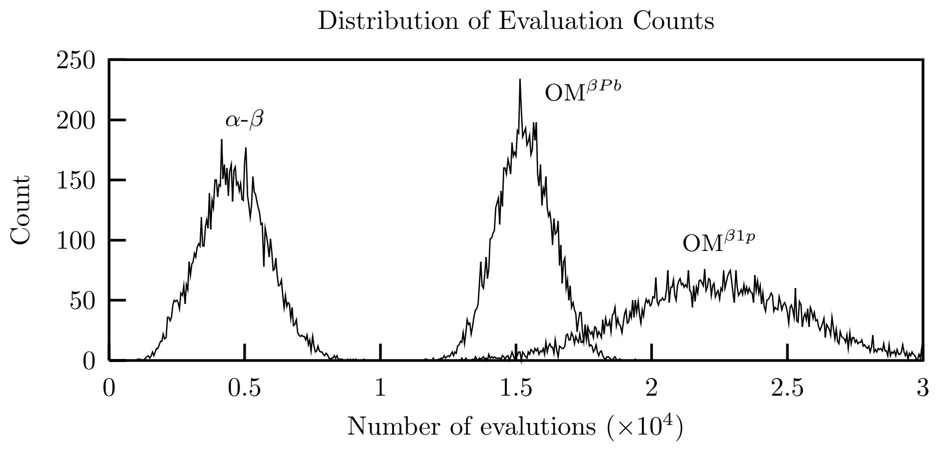


Figure 5.2: Histograms that show the distribution of the number of evaluations needed in 10,000 random game trees with depth 8 and width 4 for $\alpha$-$\beta$, OM$^{\beta 1p}$, and OM$^{\beta Pb}$.

needed compared to the number of evaluations needed by $\alpha$-$\beta$ search. Figure 5.3 shows these relative distributions for OM$^{\beta 1p}$ and OM$^{\beta Pb}$ on the same population of random game trees. The figure clearly shows that the peaks of these distribution are in accordance with the ratios of 3 and 4 as mentioned above, but the distributions are heavily skewed to the right. It should be mentioned that $\alpha$-$\beta$ search used the first evaluation function (computed with the $base_0$ values) and that $\beta$-pruning OM search pruned on basis of the second evaluation function (computed with the $base_{op}$ values). Because these functions were independently distributed, the results in this graph are probably worse than in real-world situations, especially concerning the large spreading.

The relative distributions in figure 5.3 provide more information than the absolute distributions in figure 5.2 and therefore we will present relative numbers for the rest of the experiments in this section. Since the relative distributions are more skewed than the absolute ones, we will not provide the usual mean value and variance but the following parameter-free distribution information: the median, the lower and upper quartile, and the 5th and 95th percentile of every sample.

### 5.2.3 Influence of the game-tree size

In order to study the relation between the size of the search tree and the computational complexity of both algorithms in random game trees, we conducted two series of experiments. In one series we varied the depth of the search trees and in the other series we varied the branching factor of the trees. Per value of the search depth and branching factor, a sample of 100 random game trees was taken for which the number of evaluations that OM$^{\beta 1p}$ and OM$^{\beta Pb}$ needed was determined. For every random game tree we also determined the number of evaluations that $\alpha$-$\beta$ search needed.

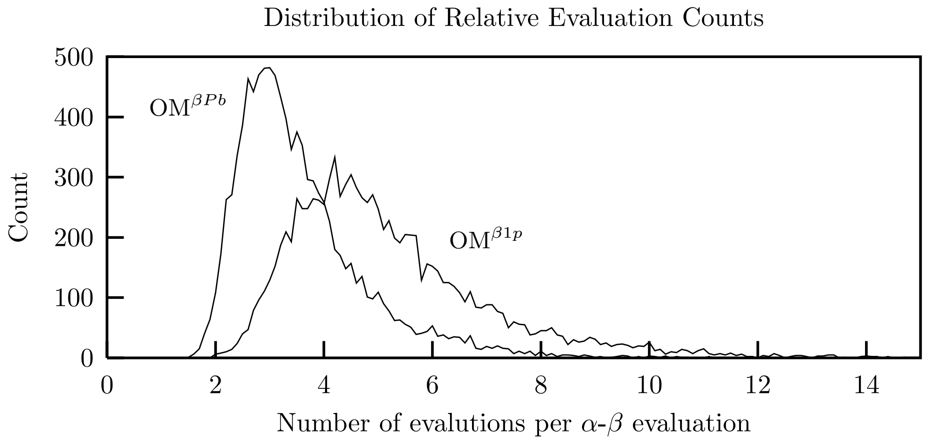


Figure 5.3: Histograms that show the distribution of the number of evaluations needed in 10,000 random game trees with depth 8 and width 4 for $\mathrm{OM}^{\beta 1p}$ and $\mathrm{OM}^{\beta Pb}$ *divided* by the number of evaluations needed by $\alpha$-$\beta$ on the same trees.

Figure 5.4 shows the results for the influence of the search depth. It shows clearly that $\mathrm{OM}^{\beta 1p}$ needed more evaluations than $\mathrm{OM}^{\beta Pb}$ on these random game trees for depths larger than 8. Furthermore, the shape of the lines suggests that the relation in the case of $\mathrm{OM}^{\beta 1p}$ is non-linear whereas $\mathrm{OM}^{\beta Pb}$ shows a more-or-less linear relation between the number of evaluations per $\alpha$-$\beta$ evaluation and the search depth.

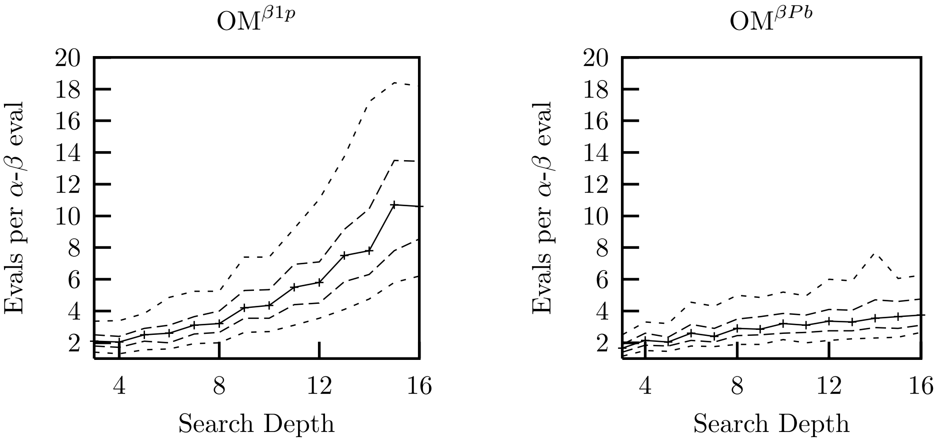


Figure 5.4: Relative number of evaluations needed by $\mathrm{OM}^{\beta 1p}$ and $\mathrm{OM}^{\beta Pb}$ as a function of the search depth. The branching factor is 3 and the sample size is 100. The solid line in the middle is the median of each sample, the inner dashed lines represent the lower and upper quartiles, and the outer dotted lines represent the 5th and 95th percentile of the samples.

The influence of the branching factor is shown in figure 5.5. Again the figure clearly shows that OM$^{\beta 1p}$ needed more evaluations than OM$^{\beta Pb}$ on these random game trees. Both algorithms show a more-or-less linear relation between the number of evaluations per $\alpha$-$\beta$ evaluation and the branching factor.

The analysis in section 3.3 showed that in the best case, OM$^{\beta Pb}$ performs worse than OM$^{\beta 1p}$, which appears to be in contrast with the average case.

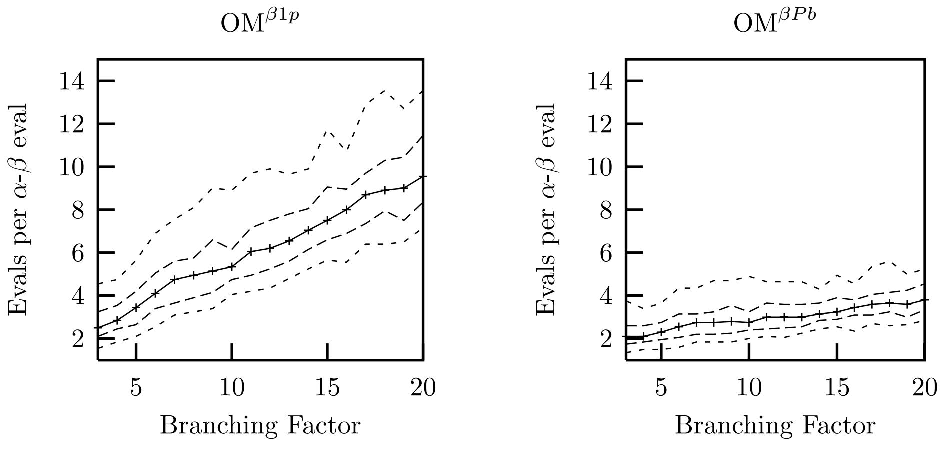


Figure 5.5: Relative number of evaluations needed by OM$^{\beta 1p}$ and OM$^{\beta Pb}$ as a function of the branching factor. The search depth is 5 and the sample size is 100. The meaning of the lines is the same as in figure 5.4.

### 5.2.4 Effect of transposition tables

As we explained in subsection 3.4.2, the $\alpha$-$\beta$ probes in the probing versions of OM search allow for the application of transposition tables. Since some parts of the search tree will be visited during more than one probe, we expect an advantage of the transposition table, even if no iterative deepening is applied. We performed an experiment to measure the effect of a transposition table on OM$^{\beta Pb}$ for different sizes of random game trees. The size of the transposition table was $2^{18}$ entries, the replacement scheme used was *deep* (see Breuker *et al.*, 1994). We did not apply iterative deepening.

Figure 5.6 shows the effect for various search depths (left) and branching factors (right). Since it is difficult to see the precise difference with the results in figures 5.4 and 5.5, we present in figure 5.7 the improvement that is caused by the transposition table. The figure shows that with increasing search depth the improvement grew to a level of 40 %. The improvement of transposition tables for higher branching factors was less pronounced than for lower branching factors.

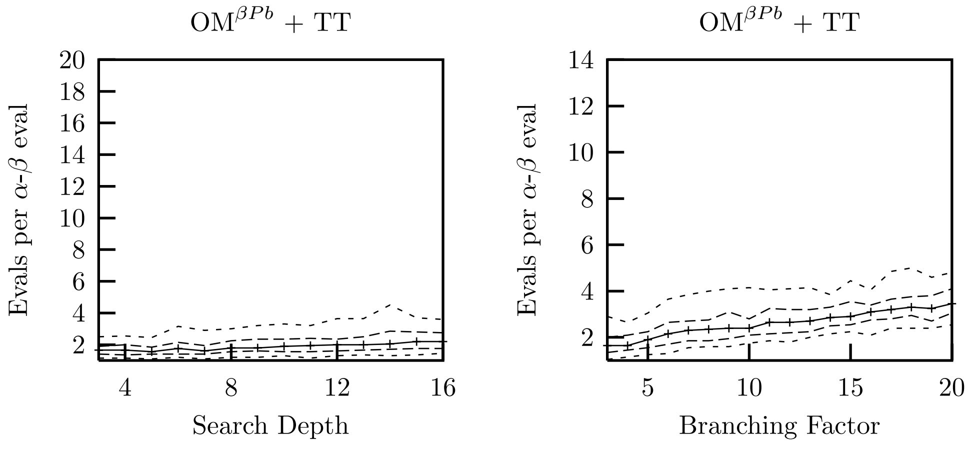


Figure 5.6: Relative number of evaluations needed by $\mathrm{OM}^{\beta Pb}$ augmented with a transposition table as a function of the search depth and branching factor. The branching factor is 3 in the left graph and the search depth is 5 in the right graph. The sample size is 100. The meaning of the lines is the same as in figure 5.4.

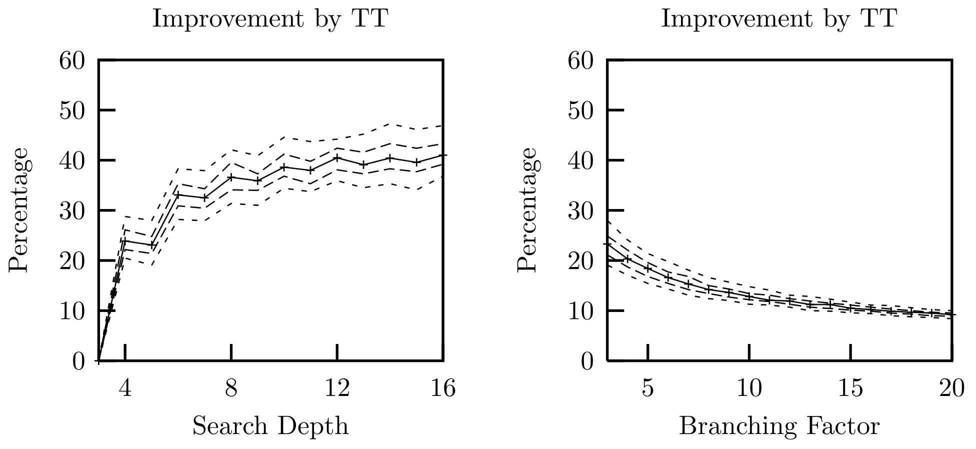


Figure 5.7: Improvement caused by a transposition table in $\mathrm{OM}^{\beta Pb}$, as a function of the search depth and branching factor. The graph shows the percentage of evaluations that is skipped when the transposition table is used.

### 5.2.5 Discussion

The experiments with random game trees indicated that $\mathrm{OM}^{\beta Pb}$ performs better in this setting than $\mathrm{OM}^{\beta 1p}$. This is in contrast with the best case in which $\mathrm{OM}^{\beta 1p}$ is the most efficient. Furthermore, $\mathrm{OM}^{\beta Pb}$ can be augmented with transposition tables. The experiments in subsection 5.2.4 showed that the improvement was considerable, even without iterative deepening.

The results of the random game trees are not easy to extrapolate to real games. There are two reasons for this. First, in these experiments there was no correlation between the evaluation functions $V_0$ and $V_{op}$. In real games the two evaluation functions will be correlated to some extent. A higher correlation between the evaluation functions will probably reduce the spreading in the distributions of figure 5.3, because when $V_0$ and $V_{op}$ are highly correlated, a move ordering that causes much or less pruning in $\alpha$-$\beta$ search will then also cause much or less pruning in OM search. Second, the random game trees have a random move ordering. The pruning in both $\alpha$-$\beta$ search and OM search will heavily profit from a better move ordering. Together this means that the computational-complexity results for the random game trees must be viewed more as a 'worst-case' scenario than an 'average case'.

## 5.3 Experiments in Lines of Action

As a first investigation of the effectiveness of OM search, we conducted tournaments in the game lines of action (LOA) between players using $\alpha$-$\beta$ search and OM search at several search depths. LOA was selected because the game is not trivial and also not too complex so that a reasonable amount of games could be played with a varying set of parameters. Furthermore, there are various evaluation functions available, which allowed for the application of OM search.

### 5.3.1 Rules of the game

LOA is a board game invented in 1960 by Claude Soucie (and described in Sackson, 1992). It is played on an $8 \times 8$ checkers board. The starting player uses twelve black stones, the other player uses twelve white stones. The black stones are placed in two horizontal rows along the top and bottom of the board, while the white stones are placed in two vertical rows at the left and right edge of the board (see figure 5.8, left). The players move in turn. A player may move only one stone at the time (i.e., per turn) in a straight line in any of the eight principal directions. The step length of the move is exactly equal to the total number of stones (black and

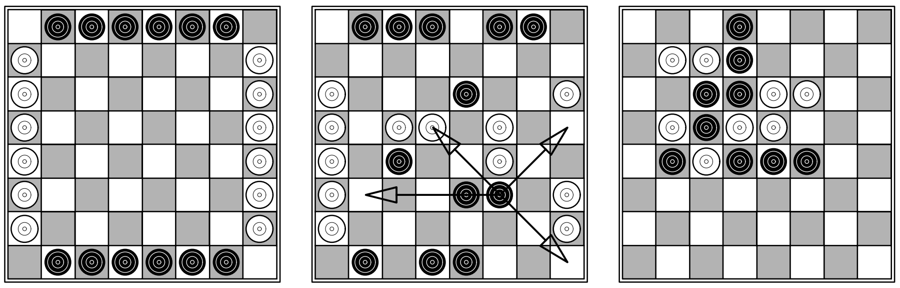


Figure 5.8: LOAexamples: the opening position, legal moves for one piece, and an end position.

white) on the line of movement (figure 5.8, middle). A player is only allowed to jump over the own stones. When the move lands on an opponent's stone, this stone is captured. The object of the game is to move the stones until all own stones are connected orthogonally or diagonally (figure 5.8, right). In the implementation used in the experiments, a repetition of a board position instantaneously ends the game (in a draw). There is some dispute on details in the rules of the game (see Winands, 2003), but these are not crucial to our research.

### 5.3.2 Evaluation functions

In the tournaments we applied three different evaluation functions. The first one ($V_c$) is a simple function that assigns values to stones depending on their board position: the four centre squares produce 10 points, their 12 surrounding squares produce 5 points, and the 20 squares surrounding the latter squares produce 1 point. The remaining 28 border squares give no points. Opponent stones receive the same amount of points (expressed as minus points). The score of a board position is the total of all points assigned to the stones. This centre-oriented evaluation function is in general only one part of the evaluation function in a LOA-playing program. Of course, in practice many more aspects are used to evaluate a position.

The two other evaluation functions that were used in the tournaments ($V_{NNa}$ and $V_{NNb}$) are neural network functions that are trained especially to evaluate board positions in LOA (Kocsis, Uiterwijk, and Van den Herik, 2001). The neural networks are feed-forward networks, consisting of three layers. The first layer (X) has 64 input nodes corresponding to the squares on the board and 1 node to indicate the player to move. The second layer (H) has 40 nodes. The third layer (Y) is a single output node, corresponding to the evaluation value. Every node is connected to all nodes in the following layers (nodes in X to nodes in H and Y, nodes in H to the node in Y). The activation level of the nodes is controlled by a sigmoid function. The networks were tuned by the weights on their 2705 connections. These weights were produced by a type of reinforcement learning (TD($\lambda$), see Sutton, 1988) during which neural-network players competed with each other in many games meanwhile learning from the own prediction errors. The search depth during the training was 3.

### 5.3.3 Set-up of the tournaments

We conducted three tournaments (A,B,C) with the following set-up. In every tournament we tested two evaluation functions (indicated here by $V_1$ and $V_2$). Each tournament consisted of an OM part and a normal part. In the OM part the first player used $\beta$-pruning OM search with $\alpha$-$\beta$ probing (taking $V_0 = V_1$ and $V_{op} = V_2$) and the second player used $\alpha$-$\beta$ search (with $V_2$). The evaluation function that the first player assumed for MIN was the same one that the second player actually used. So, the first player had perfect knowledge of the opponent. To be able to compare the results of OM search with $\alpha$-$\beta$ search, each OM part was replayed in the normal part of that tournament; now both players used $\alpha$-$\beta$ search, the first player with $V_1$, the second player with $V_2$. To investigate the influence of the search depth, every set was played using four different search depths (2, 3, 4, and 5). No time limit was given. Hence each tournament consisted of four OM matches and 4 normal matches. In each match a set of 30 different positions was played to the end. These positions are provided in appendix A. The set had to be played twice, since we reversed colours, so the total number of games was 60 per match.

### 5.3.4 Results and discussion

The outcomes of the tournaments are presented in table 5.1. The outcome of a match is given by its results; +24 means 42-18 and −12 means 24-36.

The results for OM search in tournament A are surprising. Although the opponent model was correct, the player using OM search profited only at search depth 2 from the knowledge given. At the search depths 3, 4, and 5 using OM search with the correct opponent model even led to a decrease in performance in relation to $\alpha$-$\beta$ search.

In tournament B the matches were between the evaluation functions $V_{NNa}$ and $V_c$, of which we proved in a separate tournament that $V_c$ is much weaker than $V_{NNa}$. Here the results of OM search were even more remarkable than in tournament A. A winning evaluation function (see the normal part) was transferred into a losing one by using the correct opponent model.

Tournament C was the reverse of tournament B, i.e., with $V_c$ having an opponent model of $V_{NNa}$. Hence the results of the normal part were exactly opposite to those of tournament B. However, using OM search gave analogous bad results at the depths 3, 4, and 5.

We remark that in all three tournaments, the OM player was given a serious advantage because not only perfect knowledge of the opponent was provided, but the search depth was kept constant, independent of the time that is needed to search the tree. The difference in performance between $\alpha$-$\beta$ search and OM search could not be caused by a difference in search efficiency.

An explanation for the poor behaviour of OM search in these experiments is the observation that the pairs of evaluation functions were not admissible (see section 4.2.2). The absence of admissibility can have caused type-I errors that led to bad choices for the OM player, even if perfect knowledge of the opponent was available.

Table 5.1: Results of the three LOA tournaments. The numbers indicate the result over 60 games for the first player.

| | Tournament A $V_1 = V_{NNa}$, $V_2 = V_{NNb}$ | | Tournament B $V_1 = V_{NNa}$, $V_2 = V_c$ | | Tournament C $V_1 = V_c$, $V_2 = V_{NNa}$ | |
|---|---|---|---|---|---|---|
| Search Depth | OM - $\alpha$-$\beta$ | $\alpha$-$\beta$ - $\alpha$-$\beta$ | OM - $\alpha$-$\beta$ | $\alpha$-$\beta$ - $\alpha$-$\beta$ | OM - $\alpha$-$\beta$ | $\alpha$-$\beta$ - $\alpha$-$\beta$ |
| 2 | +24 | +18 | −2 | −4 | +6 | +4 |
| 3 | −12 | +4 | −12 | +24 | −8 | −24 |
| 4 | −4 | +6 | −32 | +18 | −28 | −18 |
| 5 | −16 | −20 | −32 | +24 | −36 | −24 |

Furthermore, the quality of the evaluation functions was not high, resulting in large overestimations, thereby increasing the effect of type-I errors.

## 5.4 Experiments in the Chess Endgame KQKR

In this section we investigate to what extent the performance of OM search increases when type-I errors are prevented by using an admissible pair of evaluation functions. We also want to study the influence of restricted speculation (see subsection 3.1.4) on the performance of OM search.

An apparent way to achieve admissibility in OM search is to use a perfect evaluation function. An excellent instance of such an evaluation function is an endgame database. When the player is in a winning position that is available in the endgame database at hand, OM search is of no use, since the player should just follow the strategy as dictated by the endgame database. It is more interesting when the player is in a losing position. Now OM search can be advantageous, because the knowledge of the opponent's strategy could be used to lure the opponent in a position that the endgame database indicates as won for the player, or at least as a draw.

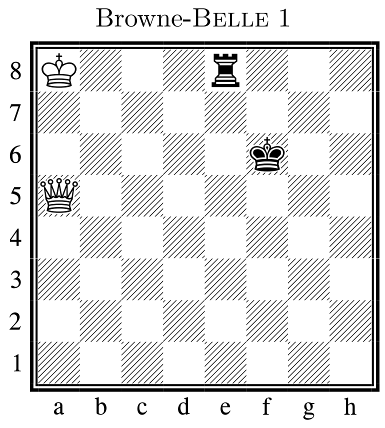
A setting that consists of a player using an endgame database at a losing position against a fallible player has been studied earlier in the domain of chess. In 1977, Ken Thompson used two well-known maximin[2] positions in the KQKR endgame (figure 5.9) to test the difficulty of the KQKR endgame in a contest with International Grandmaster Walter Browne (see Fenner, 1979; Levy and Newborn, 1991; Jansen, 1992b; Newborn, 1997; in appendix B we list the complete games as played). In these matches, Browne played the stronger side (White), but BELLE had perfect knowledge of the endgame. The first match (starting from figure 5.9 left) ended in a draw at move 45. The second match (starting from figure 5.9 right) was won by Browne at the $50^{th}$ move. Both positions are theoretically a win for White in 31 moves[3]. The properties of the KQKR endgame were analyzed in detail by Jansen (1992b).

Jansen (1993) also described experiments in the KQKR domain. In these experiments, computer programs with access to the endgame database played against human players. Jansen used three types of programs: program **R** selected randomly from the set of optimal moves, program **B** selected the best move from the set of optimal moves (with respect to the number of optimal responses by the opponent), and program **HP** (*Heuristic Program*, see subsection 2.1.4) used some knowledge on human opponents in general and was allowed to play suboptimally. The results showed that **HP** yielded better results against humans than the base case, optimal play, as executed by **R** or **B**.

We decided to conduct our experiments in the same KQKR endgame and on the same two positions. In our experiments we also used three different programs.

[2]A maximin position is a position within a given set that takes the maximal number of moves to reach a win when both sides play optimally, provided that the depth of the win is incorporated in the payoffs (see subsection 4.1.1).

[3]This (31) is the distance to conversion (capture of the Rook) of both positions, according to Thompson's database. The distance to win of both positions is 35, see appendix B.



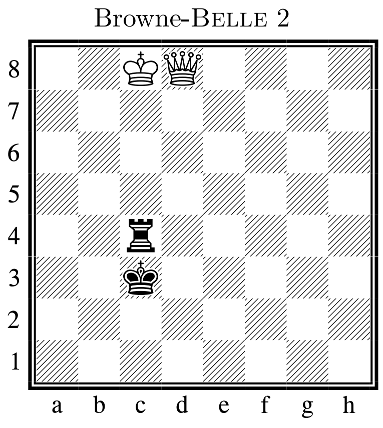


Figure 5.9: The starting positions, White to move, that Browne (White) played against the chess computer BELLE in 1977. See appendix B for the complete games.

The first program was a chess program with access to the KQKR endgame database, thus being able to play optimally. The program selected the first optimal move encountered, so it differed in this respect from both **R** and **B**. This program was our base case. The second program was the subject of our research; it was a chess program equipped with OM search, the KQKR endgame database, and perfect knowledge of the opponent. This second program was also able to apply restricted speculation. The third program was a chess program without access to the endgame database. In combination with some time restrictions, this third program acted as a fallible opponent playing the KQ-side in the endgame.

Before the experiments we formulated two hypotheses. Our main hypothesis was that, as in Jansen's experiments, OM search (playing the KR-side) would perform better than plain optimal play because the opponent program would be lured into traps that will cause it to make errors. The second hypothesis stated that the depth of the OM search's speculations on the opponent's moves would influence the performance of OM search. If the speculative search would be too shallow, OM search would not be able to find any opportunities to mislead the opponent. If the speculative search would be too deep, the predictions of the opponent's behaviour would then be based on the remaining shallow $\alpha$-$\beta$ probes of which the heuristic evaluations would no longer be sufficiently reliable and therefore would cause OM search to make wrong decisions.

## 5.4.1 Implementation

To test OM search on the Browne-BELLE chess endgame positions, we implemented the enhanced version of $\beta$-pruning OM search with probing and restricted speculation (see algorithms 3.5 and 3.6) into Hyatt's chess program CRAFTY (version 18.11) (Hyatt, 2002b). We selected this program for two reasons: (1) the source code is publicly available and (2) it is a reasonably strong chess program (it finished 7th in a field of 18 programs at the 18th World Microcomputer Chess Championship in 2001 (Schneider, 2001)).

CRAFTY's core is a negamax implementation of $\alpha$-$\beta$ search. The program is endowed with the usual search enhancements such as iterative deepening and quiescence search. Replacing its core search algorithm by OM search made some of CRAFTY's search enhancements impossible. For instance, we had to remove aspiration search because $\beta$-pruning OM search uses only $\beta$ values to prune. However, many other enhancements remained available because the probes in OM search call CRAFTY's original search routine. The remaining enhancements are transposition tables, search extensions, null moves, quiescence search, killer moves, history tables, and (Eugene Nalimov's) endgame databases[4] (The latter were disabled in the opponent player, see the beginning of this section and the next subsection.)

We made an adjustment to the root-level call of OM search: only those moves at the root level were considered that did not decrease the distance to a loss more than the single ply that the opponent could force, according to the endgame database. In other words, we did not allow OM search to play suboptimally.

### 5.4.2 Set-up of the tournaments

Three versions of the CRAFTY engine were linked together using WINBOARD: first, the base-case version (see above), which was an unchanged version of CRAFTY 18.11 with access to the KQKR endgame database; second, the version with OM search (called OMCRAFTY) and with access to the endgame database too; and, third, the version of CRAFTY 18.11 not using the endgame database. Henceforth, we denote the base-case version by OMCRAFTY with speculative search depth 0 (see below). In the experiments, OMCRAFTY used exactly the same evaluation function as CRAFTY 18.11, resulting in perfect knowledge of the opponent. During the experiments, we disabled pondering and book learning.

We performed two tournaments, one starting with the Browne-BELLE position 1 and one with position 2 (see figure 5.9). OMCRAFTY always played the weaker side (i.e., the black side, with Rook), CRAFTY without endgame database played the stronger side (with Queen). The tournaments each consisted of twelve matches. In every match, OMCRAFTY used a fixed speculative search depth, ranging from 1 to 11. As a base case, a match was played in which OMCRAFTY was replaced by CRAFTY with access to the endgame database. This match is indicated by speculative search depth 0 (see above).

All matches counted 20 games. In each game, the Browne-BELLE position 1 or 2 was played to the end (conversion, mate, or applying the 50-move rule). The total time limit for a game was 10 minutes per side, leading to a search depth of roughly 10 to 12 ply. Owing to small timing differences (Hyatt, 2002a), all games were different. Since we used restricted time per game in the chess experiments, it is important to mention the platform: a 1.33 GHz AMD Athlon PC running Windows ME. (The CRAFTY source was compiled with Microsoft Visual C++ 6.0.)

[4]Nalimov's endgame databases contain distances to win/loss, whereas Thompson's endgame databases contain distances to conversion. See also appendix B.

### 5.4.3 Results and discussion

Below we discuss simultaneously the results of both series of matches. By combining the results we are able to stress the differences. Table 5.2 gives the number of games that ended in a draw (e.g., the game length is 51). All other games ended in conversion.

Table 5.2: Number of games (out of 20) that ended in a draw due to the 50-move rule.

| | Speculative Search Depth (Ply) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Browne-BELLE 1 | 0 | 0 | 0 | 3 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 2 |
| Browne-BELLE 2 | 1 | 1 | 4 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

It should be mentioned that the search depth reached by OMCRAFTY did not exceed 11 on average in the experiments. After the sixth move of the games, the search depth of OMCRAFTY decreased to 9.5 on average. This means that the results for speculative search depths 9, 10 and 11 are largely correlated.

Figure 5.10 gives the distribution of the game length (averaged per OM-search depth) as a function of the OM-search depth. The results of the first Browne-BELLE position (left) show that using OM search increased the average game length for all values of the speculative search depth. When OM search was not used (depth 0), the games lasted 37.4 moves on average. The best performance was at OM-search depth 4, the average game length there being 42.2. Statistical analysis (one-sided

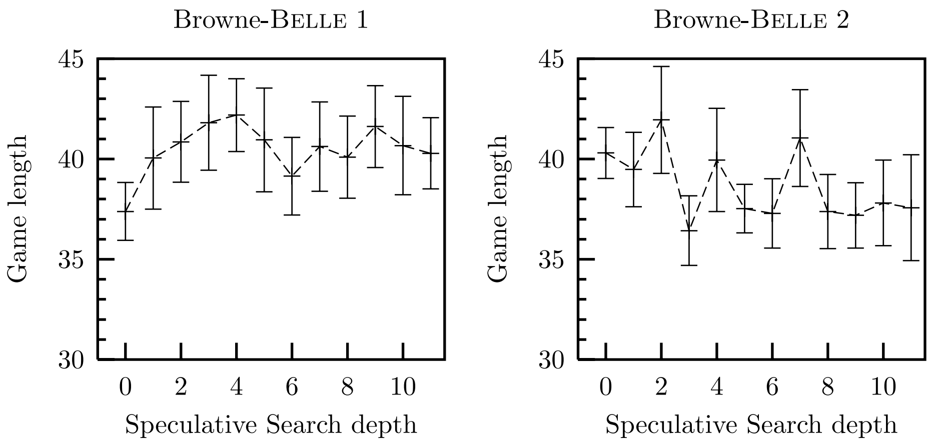


Figure 5.10: Mean game length (number of moves to conversion or draw) in the two Browne-BELLE experiments as a function of the speculative search depth. Depth 0 means that $\alpha$-$\beta$ is used instead of OM search.

*t*-test) indicates that the increase of the game length in comparison with the base case was significant (for a 95%-confidence level) for all depths except 6.

The results of the second Browne-Belle position (right) showed an opposite effect. For speculative search depth 0 (the base case) the games lasted 40.3 moves on average. For speculative search depths 3, 5, 6, 8, 9, 10, and 11 the average game length was significantly lower with a 95%-confidence level than for the base case. Since we observed that the base-case version performed better in the second position than in the first Browne-Belle position, a closer look at the experiments was needed to explain the difference in results for the two Browne-Belle positions.

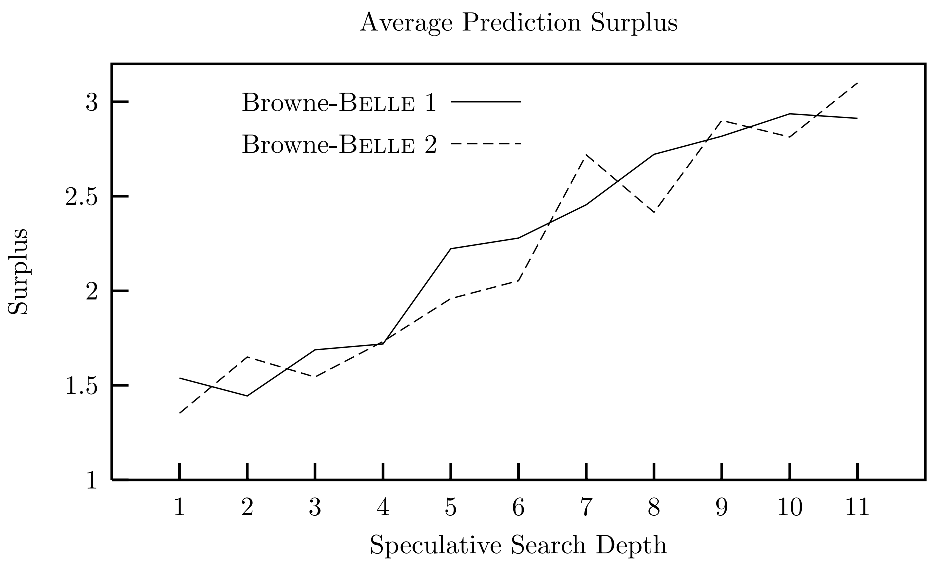


Figure 5.11: Average prediction surplus (OM-value minus endgame-database value) of the two Browne-Belle positions for varying speculative search depth.

In order to show how OM search performed during the experiments, we measured the increase of the game length as predicted by OM search, i.e., the OM-search value minus the endgame-database value. We call this increase the *prediction surplus*. In figure 5.11 the average of this prediction surplus is plotted as a function of the speculative search depth. The figure shows that the prediction surplus grew almost linearly with the speculative search depth. No significant difference between the two Browne-Belle positions was registered. This result is to be expected from OM search: the more deeply speculative search is performed, the more opportunities for misleading the opponent are expected.

Since no difference in the behaviour of OM search itself was registered between the two series of matches, the behaviour of the opponent (i.e., the opponent's error) might provide some clue. For our investigations we used the same notion of error as in Jansen (1992b): the number of moves lost with respect to optimal play. Two graphs provide some insight into the intricacies of OM search. Figure 5.12 (left)

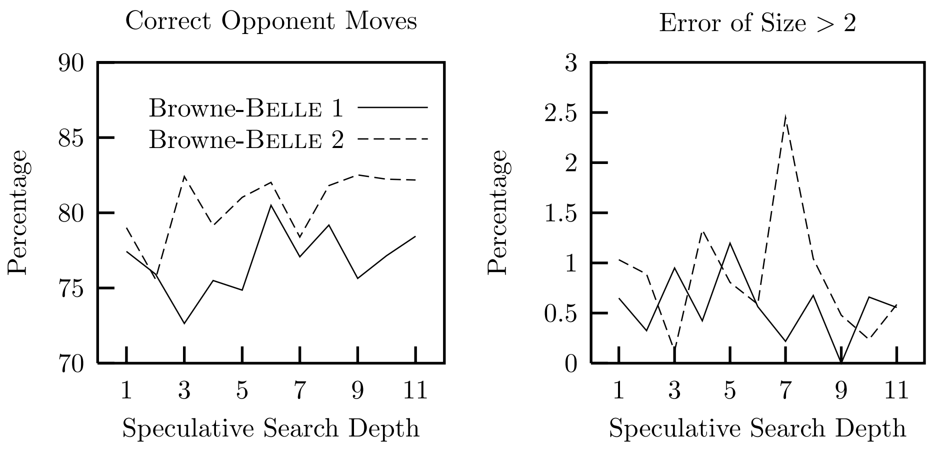


Figure 5.12: Percentage of correct opponent moves (left) and of the opponent's moves in which the opponent made an error that prolonged the distance to win with more than 2 extra moves (right).

shows the percentage of moves in which the opponent made no error, and figure 5.12 (right) shows the distribution of errors larger than 2. The tables 5.3 and 5.4 provide more detailed information on the distribution of errors of all sizes. It appears that the opponent made errors in 20 to 25 per cent of the moves. Most of these errors (about three quarters) had size 1, and only a few had a size larger than 2. The maximum error recorded had size 11, which occurred only twice.

Figure 5.12 (left) indicates that, in general, the opponent made fewer errors in the second Browne-Belle positions than in the first one. This agrees with the shorter lengths of the games in figure 5.10. The correlation is also visible in some details, e.g., the peak of the game length around speculative search depth 4 for the first position (figure 5.10, left) is related to the minimal percentage of correct moves at these depths (figure 5.12, left). The outlaying value for OM-search depth 7 at the second Browne-Belle position (figure 5.10, right) coincides with the peak of errors larger than 2 at that depth (figure 5.12, right).

Table 5.3: Distribution in percentages of the errors (there are no errors of size 8) by the opponent per speculative search depth in the experiments of the first Browne-BELLE position.

| | Error size | | | | | | | | | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Depth | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 9 | 10 | Length |
| 0 | | | | | | | | | | | 37.4 |
| 1 | 77.43 | 18.36 | 3.56 | 0.65 | | | | | | | 40.0 |
| 2 | 75.92 | 20.39 | 3.36 | 0.11 | 0.11 | 0.11 | | | | | 40.9 |
| 3 | 72.65 | 21.33 | 4.96 | 0.32 | 0.32 | 0.11 | 0.11 | 0.11 | | | 41.8 |
| 4 | 75.50 | 19.85 | 4.22 | 0.32 | | | | | 0.11 | | 42.2 |
| 5 | 74.86 | 19.91 | 4.03 | 0.65 | 0.11 | 0.22 | 0.11 | | | 0.11 | 41.0 |
| 6 | 80.50 | 15.67 | 3.27 | 0.45 | | | 0.11 | | | | 39.1 |
| 7 | 77.07 | 19.65 | 3.06 | 0.11 | | 0.11 | | | | | 40.6 |
| 8 | 79.19 | 16.20 | 3.94 | 0.67 | | | | | | | 40.1 |
| 9 | 75.64 | 20.21 | 4.15 | 0.00 | | | | | | | 41.6 |
| 10 | 77.14 | 16.92 | 5.27 | 0.33 | | 0.33 | | | | | 40.7 |
| 11 | 78.44 | 18.44 | 2.56 | 0.44 | 0.11 | | | | | | 40.3 |

Table 5.4: Distribution in percentages of the errors (there are no errors of sizes 8, 9, and 10) by the opponent per speculative search depth in the experiments of the second Browne-BELLE position.

| | Error size | | | | | | | | | Avg. |
|---|---|---|---|---|---|---|---|---|---|---|
| Depth | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 11 | Length |
| 0 | | | | | | | | | | 40.3 |
| 1 | 79.01 | 17.66 | 2.29 | 0.80 | | 0.11 | | 0.11 | | 39.5 |
| 2 | 75.58 | 19.20 | 4.22 | 0.55 | | | 0.11 | | 0.22 | 41.9 |
| 3 | 82.41 | 16.27 | 1.20 | 0.12 | | | | | | 36.4 |
| 4 | 79.16 | 15.96 | 3.55 | 0.89 | 0.33 | 0.11 | | | | 40.0 |
| 5 | 81.03 | 15.98 | 2.18 | 0.80 | | | | | | 37.5 |
| 6 | 82.02 | 15.63 | 1.76 | 0.47 | 0.12 | | | | | 37.2 |
| 7 | 78.38 | 16.19 | 2.98 | 2.45 | | | | | | 41.0 |
| 8 | 81.81 | 15.41 | 1.74 | 0.93 | 0.12 | | | | | 37.4 |
| 9 | 82.51 | 15.93 | 1.08 | 0.48 | | | | | | 37.2 |
| 10 | 82.23 | 15.40 | 2.13 | 0.24 | | | | | | 37.8 |
| 11 | 82.18 | 15.01 | 2.23 | 0.59 | | | | | | 37.6 |

Why did the opponent make fewer errors in the second Browne-Belle position? In 1977, Thompson assumed that the two positions were equally difficult, since both had a maximin distance to conversion of 31. However, in our experiments the results for the two positions were quite different. To emphasize this observation, we present in figure 5.13 the number of different variations of the game as encountered in the experiments; they are given as a function of the the number of moves made. In the experiments, almost no game developed equally. The graph shows that after 40 plies there were 180 different variations in the first Browne-Belle position, but only 80 in the second position. This indicates that the variations that started at the second position were easier to play and therefore led to fewer errors by the opponent.

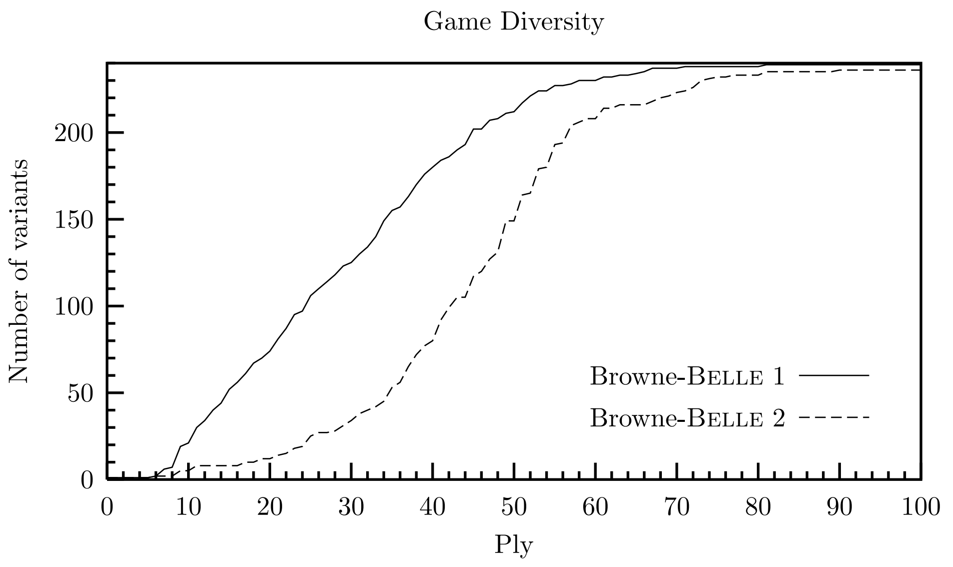


Figure 5.13: Game diversity of the two Browne-Belle positions, expressed in the number of different variations of the game occurring in the experiments.

The difference between the two positions might be caused by some effect in the evaluation function of Crafty since the first moves of all games played in our experiments were the same: (1. Kb7 Re7+) for the first Browne-Belle position and (1. Kb7 Rb4+) for the second Browne-Belle position. The resulting positions (8/1K2R3/5k2/Q7/8/8/8/8 and 3Q4/1K6/8/8/1r6/2k5/8/8) are symmetrical copies of each other, so after the first move both positions are basically equal.

To complete the discussion of the results for the chess tournament, we present some data on the computational efficiency of OM search in this setting. Figure 5.14 shows a comparison of the search depth and the effective branching factor between OmCrafty and Crafty. The effective branching factor is computed from the search depth and the number of positions that is evaluated. It can be interpreted as a measure of the amount of pruning. The figure shows that OmCrafty searched one ply less deep than Crafty in the same time. The amount of pruning, expressed by the effective branching factor, was comparable.

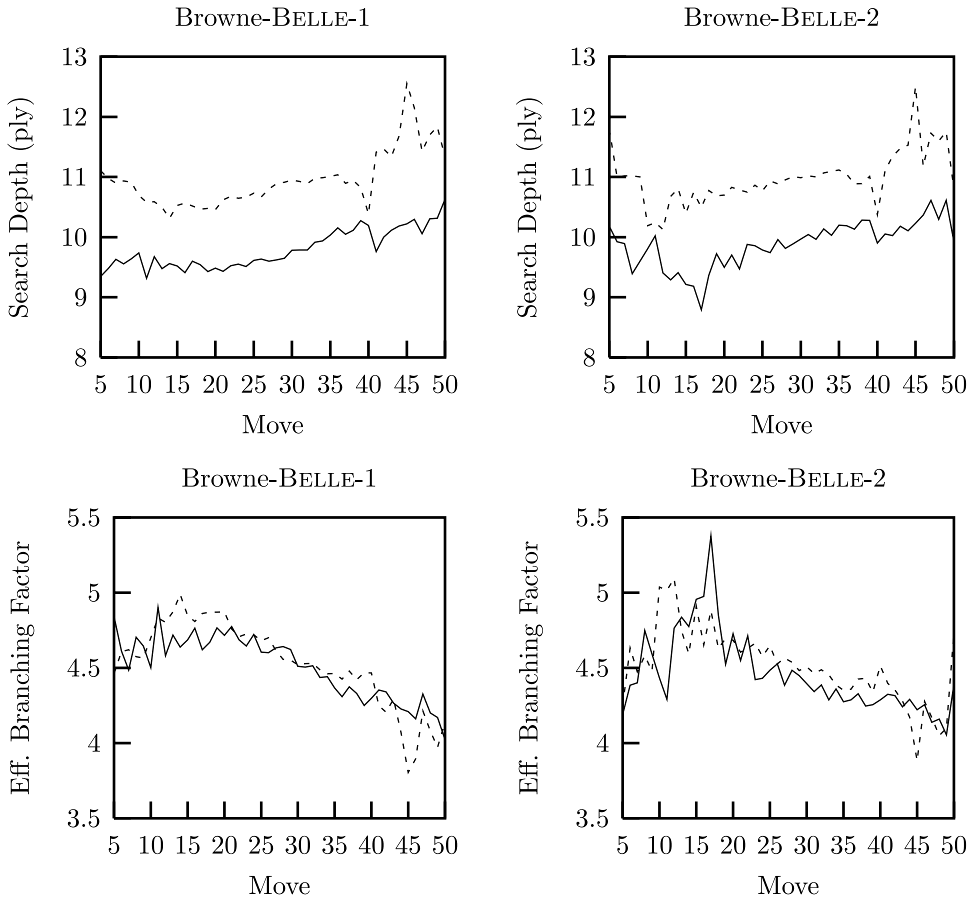


Figure 5.14: Search efficiency of Crafty (dashed line) and OmCrafty as a function of the move number.

## 5.5 Experiments in Bao

In the last series of experiments in this chapter we wanted to study under what conditions OM search can be made successful. We provided the OM-search player with all the resources needed for this, among them a larger search depth. In contrast to the previous experiments in the chess endgame, we did not use admissible pairs of evaluation functions. To lower the risk of type-I errors, we allowed the OM-search player only 1 ply of speculative search.

The experiments were performed on a game called bao. Since this is a fairly unknown game, we provide some more information on the background of the game. We also go into more detail on the generation of a set of evaluation functions that we use in the experiments. In subsection 5.5.1 we first give an introduction to the family of mancala games to which bao belongs. In subsection 5.5.2, the properties of bao are discussed. In subsection 5.5.3 we explain how we obtained the five evaluation functions for bao. Subsection 5.5.4 gives the tournament set-up and in the last subsection, 5.5.5, we present and discuss the results.

### 5.5.1 Mancala games

In large parts of the world, for a long time board games of the mancala group have been played in different versions (Murray, 1952; Russ, 2000). Most mancala games share the following properties:

- the board has a number of *holes*, usually ordered in rows;
- the game is played with indistinguishable *counters* (pebbles, seeds, shells);
- players own a fixed set of holes on the board;
- 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 capture;
- the goal is to capture the most counters.

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, wari, awele, bao, dakon, and pallankuli). For an overview and rules of many mancala games, we refer to Russ (2000).

Until recently, the western world has had little experience with mancala games. In the Middle Ages, Arabic traders might have shown a game called manqala to their European customers, but only vague references exist of Europeans playing the game. In later times, African slaves played wari in the Caribbean, but in the United States, slaves were forbidden to play their traditional games. European colonists in Eastern Asia and India must have observed mancala games being played, but apparently they were not interested enough to bring it back to Europe (Murray, 1952).

Recently, Western interest in mancala games increased. In the 1950s William J. Champion invented the game kalah (1963). It is a relatively simple version of a mancala game intended as an educational game for children. Although the game was immediately commercialised, it only became widely known when the International Red Cross adopted the game and disseminated it. In the USA the game of kalah reached the pubs and in Europe African immigrants took the game awale with them. Nowadays, several mancala games are played on the Mind Sports Olympiad (wari, omweso, and bao)[5]. Much information on mancala games is available on the Internet and there exist many computer programs for playing mancala games. Even some mobile phones are provided with them.

Mancala games have a relatively long track in artificial-intelligence research on games. Kalah has been investigated a decade after its invention as an example of game programming (Russel, 1964; Bell, 1968; Slagle and Dixon, 1970). The game of kalah has been solved in 2000 (Irving, Donkers, and Uiterwijk, 2000). Awari has been studied since the early 1980s (Van der Meulen, Allis, and Van den Herik, 1990)

[5]See http://www.msoworld.com.

and has been played in the Computer Olympiads for some time. In the research on awari, much effort is put on the construction of endgame databases (Allis, Van der Meulen, and Van den Herik, 1991; Lincke and Marzetta, 2000; Van der Goot, 2001). This has led to the complete solution of awari in May 2002 (Romein and Bal, 2002).

### 5.5.2 Properties of bao

Among the mancala games, (Zanzibar) bao is regarded as the most complex one (De Voogt, 1995). This is mainly owing 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 the expensive boards and official tournaments are organized.

The exact rules of the game are given in appendix C. Below, we will 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. 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 on 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 dictated by deterministic rules.

If a capture is possible then it is obliged in bao. This 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. These counters are immediately sown in the own front row. It implies that the game does not converge like kalah and awari.

Moves are composite. If at the end of a sowing, a capture is possible, the captured counters are sowed immediately at the own side of the board. This second sowing can again result in a new capture followed by a new sowing. 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 goes on until an empty hole is reached.

Moves can be endless because in a non-capture move, sowing can go on forever. The existence of endless moves can be proven theoretically (Donkers, Uiterwijk, and De Voogt, 2002). In real games, moves that take more than an hour of sowing also occasionally happen, but players usually make small mistakes during sowing or simply quit the game, so real endless moves never lead to endless sowing.

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 this 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 the second 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. We note that the goal of bao is not to capture the most stones, but to immobilize the opponent.

In Donkers and Uiterwijk (2002), an analysis of the game properties of bao is provided. The state-space complexity of bao is approximated to be $1.0 \times 10^{25}$, which is much higher than those of awari ($2.8 \times 10^{11}$) and kalah ($1.3 \times 10^{13}$). The shortest game possible takes 5 ply, but most games take between 50 and 60 ply because they end soon after the start of the second stage. The maximum number of moves possible at any position is 32, but the average number of possible moves varies between 3 and 5, depending on the stage of the game. 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 that has to be traversed during search ($w^d$). For bao the estimate is roughly $10^{34}$. This number together with the game-tree complexity ($10^{25}$) places bao in the overview of game complexities above checkers and in the neighbourhood of qubic (Van den Herik, Uiterwijk, and Van Rijswijck, 2002).

### 5.5.3 Generating evaluation functions

In order to conduct the OM-search experiments, we created 5 different evaluation functions. We wanted these evaluation functions to have an increasing order of operational quality, in the sense of ordering 4 in subsection 4.1.2.

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 competitive bao-playing program[6] and is therefore called DEFAULT. This function incorporates some rudimental strategic knowledge of bao. For instance, it is good to have more stones in the back row since this increases the mobility in the second stage of the game. 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 (a special hole, see appendix C) 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 (Holland, 1975). 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, one gene per hole in the front row for capturing, one gene for an active house, and another gene for capturing the opponent's house. The total score of a position was 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 $\alpha$-$\beta$ search with search depth 6. 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 per cent for large changes in a gene (i.e., generating a new random number for the gene) and 20 per cent 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

[6] Available at http://www.cs.unimaas.nl/∼donkers/games/bao.

was the winner of the previous run. The name of the resulting evaluation function is GA3.

Thereafter we used another machine-learning technique to create the fourth evaluation function, namely TD-Leaf learning (Baxter *et al.*, 1998). This is a temporal-difference learning method that is specialized 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 that was 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 $\lambda$-factor and the annealing factor both were set to 0.99. This produced our fourth evaluation function, called TDL2B.

The last evaluation function was also produced by TD-Leaf learning, but this time we used a *normalized Gaussian network* (NGN) as evaluation function, similar to the way in which Yoshioka, Ishii, and Ito (1999) trained an evaluation function for the game othello. 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 won out of 25 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 $\lambda$-factor was set to 0.8 and the annealing factor was set to 0.993. The last evaluation function is called NGND6A.

## 5.5.4 Set-up of the tournaments

We conducted seven different 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. Each match between two players consisted of 100 games; hence each tournament counted 2000 games. These games began at the start positions given in appendix D and were played to the end. To prevent problems with infinite moves, any move that involved the sowing of more than 100 stones was considered infinite and therefore illegal. A position at which a player could perform only one of these long moves was a loss for that player.

In the first two tournaments, both players used $\alpha$-$\beta$ search. In the other five tournaments, South used OM search with perfect knowledge of the opponent's evaluation function. North always used $\alpha$-$\beta$ search with search depth 6. The search depth of South differed per tournament. No time restriction was given. We used the $\beta$-pruning implementation of OM search with $\alpha$-$\beta$ probes, and allowed only one ply of speculation. Since in bao draws are not possible, and since we aimed to compare the performance of the different search algorithms used by South, the score of a match was just the number of games out of 100 that was won by South.

At every position at which South was to move, we also detected the move(s) that $\alpha$-$\beta$ search would select for South. In this way we were able to count the number of times that OM search differed from $\alpha$-$\beta$ search.

In the implementation of the $\alpha$-$\beta$ probes for OM search we took care of the fact that (some of) the evaluation functions are asymmetric. The asymmetry implies 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. Furthermore, we implemented the part of the enhanced version of OM search (see section 2.3.2) that deals with multiple equipotent moves for MIN: if MIN has multiple equal choices, MAX would select the move with the lowest value for $v_0$.

The exact set-up of each of the seven tournaments will be explained along with the results in the next subsection.

### 5.5.5 Results and discussion

#### First tournament: $\alpha$-$\beta$ plain

Table 5.5 gives the outcome of the first tournament. Both South and North used $\alpha$-$\beta$ search with search depth 6. The table clearly shows that the evaluation functions differ in quality and that every evaluation function is indeed operationally better than any of the evaluation functions above them in the table.

#### Second tournament: $\alpha$-$\beta$ extended

The second tournament was a checking tournament. South was allowed to search two extra plies (8 instead of 6). The results are presented in table 5.6. The table shows that all players profited from the increased search depth. Only the match of DEFAULT against GA3 was less fortunate for DEFAULT. This illustrates the poor quality of this evaluator.

#### Third tournament: OM plain

In the third tournament, South used OM search with one ply of speculation (OM-search depth 1) and with search depth 6. The results in table 5.7 show that 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.

Table 5.5: Results of the first tournament between 5 evaluation functions for bao. 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 | NGND6 | Score |
|---|---|---|---|---|---|---|
| MATERIAL | - | 55 | 35 | 19 | 18 | 127 |
| DEFAULT | 48 | - | 54 | 30 | 28 | 160 |
| GA3 | 55 | 61 | - | 36 | 30 | 182 |
| TDL2B | 69 | 65 | 57 | - | 39 | 230 |
| NGND6A | 79 | 73 | 75 | 60 | - | 287 |

Table 5.6: Results of the second tournament between 5 evaluation functions for bao. Both sides use $\alpha$-$\beta$, but South searches 2 ply deeper (8) than North (6).

| S \ N | Material | Default | Ga3 | Tdl2b | Ngnd6 | Score |
|---|---|---|---|---|---|---|
| Material | - | 57 | 62 | 40 | 32 | 191 |
| Default | 71 | - | 52 | 49 | 34 | 206 |
| Ga3 | 80 | 75 | - | 62 | 49 | 266 |
| Tdl2b | 86 | 76 | 69 | - | 57 | 288 |
| Ngnd6a | 88 | 76 | 80 | 70 | - | 314 |

Table 5.7: Results of the third tournament between 5 evaluation functions for bao. South uses OM search with perfect knowledge of the opponent's evaluation function. The search depth is 6 for both sides, the OM-search depth is 1.

| S \ N | Material | Default | Ga3 | Tdl2b | Ngnd6 | Score |
|---|---|---|---|---|---|---|
| Material | - | 57 | 50 | 30 | 24 | 161 |
| Default | 46 | - | 46 | 26 | 25 | 143 |
| Ga3 | 59 | 57 | - | 40 | 35 | 191 |
| Tdl2b | 78 | 64 | 60 | - | 46 | 248 |
| Ngnd6a | 71 | 58 | 66 | 61 | - | 256 |

### Fourth tournament: OM extended

South was using OM search with one ply of speculation as in the third tournament, but in this tournament it was allowed to search two ply deeper. The $\alpha$-$\beta$ probes were still restricted to depth 6. This means that South had better knowledge over the game than North, a situation that is comparable to the second tournament. Table 5.8 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 allowed to search just 6 ply deep, only Material played better than in the second tournament. This indicates that searching deeper for yourself in OM search is not sufficient for success.

### Fifth tournament: OM with perfect opponent prediction

The fifth tournament gave South a different advantage: it was allowed to extend the $\alpha$-$\beta$ probes to depth 7. The search depth (for the own evaluation) was 6. In this way, South not only had perfect knowledge of the opponent's evaluation function, but South could also predict almost perfectly what North would be doing in the next move. The search depth of the $\alpha$-$\beta$ probes (which was 6, because the probes started at depth 1) was namely exactly the same as the search depth of North. In the case of equal evaluated moves, South selected the move with the lowest own evaluation. This was not necessarily the move that North would play. Table 5.9 gives the results of this tournament. All players, except Default profited from this advantage, and

Table 5.8: Results of the fourth tournament between 5 evaluation functions for bao. South uses OM search with perfect knowledge of the opponent's evaluation function. The search depth is 8 for South, with $\alpha$-$\beta$-probes to depth 6, and the search depth is 6 for North. The OM-search depth is 1.

| S \ N | MATERIAL | DEFAULT | GA3 | TDL2B | NGND6 | Score |
|---|---|---|---|---|---|---|
| MATERIAL | - | 60 | 64 | 49 | 39 | 212 |
| DEFAULT | 63 | - | 47 | 44 | 41 | 195 |
| GA3 | 70 | 66 | - | 57 | 40 | 233 |
| TDL2B | 80 | 69 | 70 | - | 56 | 275 |
| NGND6A | 84 | 68 | 71 | 59 | - | 282 |

Table 5.9: Results of the fifth tournament between 5 evaluation functions for bao. South uses OM search with perfect knowledge of the opponent's evaluation function. The search depth is 6 for both sides, but South uses $\alpha$-$\beta$ probes to depth 7. The OM-search depth is 1.

| S \ N | MATERIAL | DEFAULT | GA3 | TDL2B | NGND6 | Score |
|---|---|---|---|---|---|---|
| MATERIAL | - | 59 | 57 | 31 | 27 | 174 |
| DEFAULT | 50 | - | 48 | 32 | 23 | 153 |
| GA3 | 53 | 64 | - | 36 | 30 | 183 |
| TDL2B | 69 | 73 | 66 | - | 40 | 248 |
| NGND6A | 77 | 82 | 77 | 63 | - | 299 |

played better than in tournament 1, albeit less good than in the second tournament in which they just searched deeper. The advantage also gave less good results than the advantage in tournament 4, except for player NGND6A. From these results we can infer that knowing exactly the moves of the opponent does not help if the own judgement is too weak.

### Sixth tournament: OM perfect

The sixth tournament combined the advantages of the fourth and fifth tournament for South. The search depth for the own evaluation was 8 for South and the $\alpha$-$\beta$ probes for the opponent extended to depth 7. The results in table 5.10 show that the power of South was significantly increased. 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 less than in tournament 2.

### Seventh tournament: OM with strict risk management

In the seventh and last tournament, South applied OM search with strict risk management. South only deviated from the strategy that $\alpha$-$\beta$ search imposed if the move that OM search advised had the same Minimax value. The search depth was

Table 5.10: Results of the sixth tournament between 5 evaluation functions for bao. South uses OM search with perfect knowledge of the opponent's evaluation function. The search depth is 8 for South, with $\alpha$-$\beta$ probes to depth 7, and the search depth is 6 for North. The OM-search depth is 1.

| S \ N | MATERIAL | DEFAULT | GA3 | TDL2B | NGND6 | Score |
|---|---|---|---|---|---|---|
| MATERIAL | - | 76 | 69 | 54 | 58 | 257 |
| DEFAULT | 59 | - | 66 | 48 | 46 | 219 |
| GA3 | 75 | 77 | - | 56 | 55 | 263 |
| TDL2B | 79 | 88 | 83 | - | 57 | 307 |
| NGND6A | 80 | 88 | 85 | 68 | - | 321 |

Table 5.11: Results of the seventh tournament between 5 evaluation functions for bao. South uses OM search with perfect knowledge of the opponent's evaluation function and strict risk management. The search depth is 6 for both sides. The OM-search depth is 1.

| S \ N | MATERIAL | DEFAULT | GA3 | TDL2B | NGND6 | Score |
|---|---|---|---|---|---|---|
| MATERIAL | - | 66 | 49 | 32 | 33 | 180 |
| DEFAULT | 49 | - | 45 | 37 | 31 | 162 |
| GA3 | 63 | 62 | - | 35 | 33 | 193 |
| TDL2B | 72 | 66 | 64 | - | 46 | 248 |
| NGND6A | 75 | 71 | 76 | 69 | - | 291 |

equal to the third tournament. Since it occurred relatively often in bao that multiple moves at the same position had the same Minimax value, South did have some room to speculate. The results in table 5.11 show that this approach was successful too. All players performed better than in the first tournament. They also all performed better than in the third tournament (or equally poor in case of TDL2B).

### A summary

Table 5.12 summarizes the results of the seven tournaments. Each cell contains the total score of a tournament (400 games, playing South). 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 indeed is as follows: (MATERIAL, DEFAULT) < GA3 < TDL2B < NGND6A. The ordering of MATERIAL and DEFAULT is unclear, but both evaluation functions are poor. The table shows that if only the search depth was increased (4: OM extended) or only the prediction of the opponent was improved (5: OM perf. opp.), the results were not as good as just using $\alpha$-$\beta$ search with two additional ply of search. When both methods were combined, (6: OM perfect), the results were better. Furthermore, the table shows that using OM search with strict risk management (7: OM no risk) led to better results than using plain OM search and plain $\alpha$-$\beta$ search.

Table 5.12: Overview of the seven bao tournaments. Column $d$ indicates the search depth used by South and column $p$ indicates the depth of the $\alpha$-$\beta$ probes.

| Tournament | $d$ | $p$ | MATERIAL | DEFAULT | GA3 | TDL2B | NGND6A |
|---|---|---|---|---|---|---|---|
| 1: $\alpha$-$\beta$ plain | 6 | - | 127 | 160 | 182 | 230 | 287 |
| 2: $\alpha$-$\beta$ extended | 8 | - | 191 | 206 | 266 | 288 | 314 |
| 3: OM plain | 6 | 6 | 161 | 143 | 191 | 248 | 256 |
| 4: OM extended | 8 | 6 | 212 | 195 | 233 | 275 | 282 |
| 5: OM perf. opp. | 6 | 7 | 174 | 153 | 183 | 248 | 299 |
| 6: OM perfect | 8 | 7 | 257 | 219 | 263 | 307 | 321 |
| 7: OM no risk | 6 | 6 | 180 | 162 | 193 | 248 | 291 |

Table 5.13: Overview of the average number of moves per game in which the move that OM search selects differed from the move that $\alpha$-$\beta$ search (with search depth 6) suggested. The standard deviation ranges between 0.5 and 2.5. The average number of moves per game for South is 19.8 over all games in the tournaments.

| Tournament | MATERIAL | DEFAULT | GA3 | TDL2B | NGND6 |
|---|---|---|---|---|---|
| 3: OM plain | 3.27±2.1 | 3.36±1.8 | 2.38±1.8 | 2.47±1.7 | 2.21±1.6 |
| 4: OM extended | 4.85±2.5 | 5.43±2.3 | 4.90±2.4 | 4.46±2.3 | 4.55±2.3 |
| 5: OM perf. opp. | 3.30±1.9 | 3.12±1.8 | 2.52±1.6 | 2.43±1.6 | 2.26±1.6 |
| 6: OM perfect | 4.35±2.1 | 5.14±2.5 | 4.47±2.2 | 3.93±1.9 | 3.98±1.9 |
| 7: OM no risk | 1.83±1.5 | 0.82±1.0 | 0.29±0.5 | 0.73±0.9 | 0.50±0.7 |

### Deviations

The last overview, in table 5.13, provides insight into the number of times that OM search deviated from the $\alpha$-$\beta$-search strategy. The table shows that searching more deeply for the own evaluation had a larger effect than searching more deeply for the prediction of the opponent. The table also shows that the number of deviations was larger in tournament 4 than in tournament 6. Since the results of tournament 4 were less good than the results of tournament 6, it seems that incorrect prediction of the opponent led to extra deviations that did not contribute to a positive outcome.

## 5.6 Chapter Conclusions

In section 5.2, we studied the efficiency of two OM-search implementations on random game trees: $\mathrm{OM}^{\beta 1p}$ and $\mathrm{OM}^{\beta Pb}$. The second algorithm appeared to be the most efficient one, in contrast to the best-case situation in which $\mathrm{OM}^{\beta 1p}$ is more efficient. Moreover, the efficiency of $\mathrm{OM}^{\beta Pb}$ can be increased by using transposition tables. Depending on the branching factor and search depth, $\mathrm{OM}^{\beta Pb}$ takes on the average between 2 and 4 times the amount of evaluations that $\alpha$-$\beta$ search needs on the same tree. In most cases this will be equivalent to the cost of at most one ply of search in the same amount of time.

The experiments with lines of action in section 5.3 illustrated what can happen if OM search is used with evaluation functions that are of poor quality and that do not obey the demand of admissibility: using OM search, even with perfect knowledge of the opponent, performs worse than $\alpha$-$\beta$ search.

In the chess experiments of section 5.4, admissibility was enforced because the player that uses OM search had access to an endgame database, which acted as a perfect evaluation function. The result for OM search, however, were still not convincing. In the experiments on the first position, OM search yielded slightly better results than $\alpha$-$\beta$ search, but the experiments on the second position showed for the most cases a worse result for OM search than for $\alpha$-$\beta$ search. The relation between the OM-search depth and the results for OM search is not clear and does not provide any clues for the results of these experiments. The efficiency of OM search in the chess setting was good: it again (as in the random game trees) costs only one ply of search depth.

The results for the bao experiments in section 5.5 gave far more insights into the working of OM search. It appears that a combination of good opponent prediction and extended search depth was needed for good results. Of these two factors, the extended search depth seemed to be more important than the good prediction. The quality of the evaluation functions appeared to be important for the effect of OM search. Because the evaluation functions did not obey the admissibility demand, the results for plain OM search were not good for most of the players. With additional resources, however, OM search was made successful.

To summarize, in our experiments there were nine factors that influenced the performance of OM search with perfect knowledge of the opponent: (1) the extent of the knowledge of the opponent's evaluation function (the more knowledge the better), (2) the extent of the knowledge of the opponent's search depth (the more knowledge the better), (3) the quality of both evaluation functions (the higher quality the better), (4) the difference in quality of the evaluation functions (the larger difference the better), (5) the existence of admissibility of the pair of evaluation functions (admissibility is a pre), (6) the quality of the prediction of the opponent (the higher quality the better), (7) the quality of the prediction of the own moves (the higher quality the better), (8) the (effective) branching factor (the larger the better), and (9) the search depth (the effect is ambiguous).

# Chapter 6

# Probabilistic Opponent-Model Search

*Obtaining perfect knowledge of an opponent's strategy is a severe demand in itself. And even if the knowledge is available, Opponent-Model search is not easily made successful. In this chapter we propose a new approach: Probabilistic Opponent-Model search (PrOM search). The approach is based on a probabilistic opponent model that incorporates the player's uncertainty about the opponent's strategy, which possibly diminishes the disadvantages of OM search. We will first study the theoretical properties of the method and then investigate how it can be implemented efficiently. Finally we will study how probabilistic opponent models can be learned from observing the opponent.*

---

**Chapter contents:** The probabilistic opponent model is introduced in section 6.1. In section 6.2, we formalize a search method based on this probabilistic opponent model, called PrOM search. We then study some related methods and some of its properties. Section 6.3 is dedicated to the basic implementations of PrOM search and to implementations with pruning. Further search enhancements are studied in section 6.4. In section 6.5 we give a best-case analysis of the pruning algorithms. Variants of opponent-model learning are discussed in section 6.6. The chapter ends with some conclusions in section 6.7.[1]

[1]Parts of this chapter have been published in Donkers, H.H.L.M., Uiterwijk, J.W.H.M., Herik, H.J. van den (2001), Probabilistic Opponent-Model Search, *Information Sciences*, Vol. 135, Nos. 3–4, pp. 123–149.

## 6.1 A New Opponent Model

The experiments in chapter 4 illustrated the two main disadvantages of OM search: (1) MAX must predict the opponent's move as precise as possible, and (2) MAX must avoid to be trapped in its own overestimated positions.

The first disadvantage is caused by the strict opponent model that is used in OM search: MAX is assumed to have perfect knowledge of MIN's strategy (i.e., the evaluation function, search depth, and move order used by MIN). Perfect knowledge is often not available in which case OM search will be applied with an incorrect opponent model. If the opponent model is not correct, the predictions of the opponent's moves can be wrong.

However, although MAX may not have perfect knowledge of the opponent, MAX may have knowledge of the opponent to some degree of certainty. Our proposal is to represent MAX's uncertainty in the opponent's model by changing the fixed opponent model of OM search into a probabilistic model. This model *approximates* the opponent's strategy by using a probabilistic composition of strategies known to MAX.

### 6.1.1 Mixed strategy

Our proposed probabilistic opponent model is based on the game-theoretical concept of a *mixed strategy*. In chapter 1 we defined a strategy for a player as a function that assigns a move to every history in the game after which the player is to move. Such a deterministic strategy is called a *pure* strategy in game theory. A mixed strategy $\sigma$ is a set of pure strategies $\{s_1, \ldots, s_n\}$ together with a probability distribution $\{\Pr(s_1), \ldots, \Pr(s_n)\}$ over these strategies. A player that follows a mixed strategy, first throws a dice according to the probability distribution in order to select a pure strategy and then plays accordingly to this selected pure strategy.

In game theory, players that follow a mixed strategy are assumed to select a pure strategy *before* the game is actually played. This is in contrast with computer (and human) game-playing in which the strategy is determined during actual playing by means of game-tree search. If a player in this context would throw a dice before the start of the game and would stick to the selected pure strategy, the opponent would be able to deduce this strategy from the player's moves during the game and could adapt the own strategy accordingly. The execution of a mixed strategy for computer game-playing must therefore be stated differently in order to obtain the same results as in game theory. Instead of assuming that a player only throws a dice before the game starts, the player is now assumed to throw a dice at *every turn* to select the strategy for the next move.

Throwing a dice at every turn might seem a fundamental change. However, it is in accordance with the game-theoretic result that a mixed strategy is equivalent to a behaviour strategy in games of perfect information. A *behaviour* strategy $b_p$ for a player $p$ is a function that assigns a probability distribution over the set of moves $m(h)$ to all histories $h$ where player $p$ is to move. A player that follows a behaviour strategy throws a dice at every turn according to this probability distribution and plays the selected move. For example, both *-Minimax and the Heuristic Program of Jansen described in chapter 3 involve behaviour strategies.

Any mixed strategy $\sigma$ can be transformed into an equivalent behaviour strategy $b$ by the following equation:

$$b(m_j|h) = \Pr(m_j|h,\sigma) = \sum_{s_i \in \sigma \text{ and } s_i(h)=m_j} \Pr(s_i) \tag{6.1}$$

In games of perfect information, the best response to a behaviour strategy (and therefore also to a mixed strategy) is a pure strategy that can be determined as follows (similar to the procedure in subsection 1.2.3). Assume that MIN is using a behaviour strategy with move probabilities $\Pr(m_j|h)$. For leaf nodes, the strategy for MAX is trivial because nothing is to be selected and its value is given by the value function of the game. For internal nodes, the strategy is to first determine the value of all direct child nodes. The value of min nodes $h$, is equal to the expected payoff: $v(h) = \sum_j \Pr(m_j|h)\, v(h+m_j)$. No move has to be selected here, since we describe the strategy for MAX. At max nodes, the strategy is to select a move that leads to a child node with the highest value. The value of the max node is the value of the selected child node.

## 6.1.2 Including oneself

The second disadvantage of OM search mentioned in section 6.1 is the possibility of MAX to get trapped in overestimated positions. As we explained in chapter 4, this effect will not easily occur in Minimax search because in Minimax the opponent model is equal to MAX's strategy which causes the overestimated positions to be filtered out at min nodes. A way to repair this effect of OM search might be to make MAX's own strategy part of the opponent model. In our new opponent model, the evaluation function of MAX is therefore included as one of the pure strategies. An advantage of this approach is that possible overestimations are accounted for, a disadvantage is that it increases the complexity of the algorithm.

## 6.1.3 A probabilistic opponent model

All considerations above lead to the following probabilistic opponent model for MIN. It consists of a set of $n$ *opponent types*, $\omega_0 \ldots \omega_{n-1}$, together with a probability distribution $\Pr(\omega_i)$ over these types. Each opponent type $\omega_i$ is a Minimax player that is characterized by an evaluation function $V_i(\cdot)$. The evaluation function of the first opponent type ($V_0(\cdot)$) is the same evaluation function that MAX is using.

MIN's strategy is a mixed strategy which consists of the Minimax strategies of the $n$ opponent types and the corresponding probabilities $\Pr(\omega_i)$. At every min node, MIN randomly picks one opponent type according to the probabilities $\Pr(\omega_i)$ and plays the move that is imposed by the Minimax strategy of that opponent type. All opponent types are assumed to use the same move ordering (or to select randomly from equipotent moves) and to use the same set of terminal positions $E$, although these conditions can be relaxed.

This probabilistic opponent model must be interpreted as MAX's approximation of MIN's true behaviour, which is unknown to MAX. MAX has only explicit knowledge of a number of opponent types, based on evaluation functions, including the own one. The probabilities on these opponent types represent MAX's subjective belief that the true opponent is using that strategy. (A similar probabilistic opponent model is used by Carmel and Markovitch (1999) in the context of multi-agent systems.)

One simple example of a probabilistic opponent model is the case in which $n = 2$, and $V_1(\cdot) = \mathit{Random}()$. This example implies that MAX beliefs that MIN is using $V_0(\cdot)$, like herself, but sometimes (with a probability of $\Pr(\omega_1)$) makes a random selection at every move.

## 6.2 The PrOM-Search Method

When the procedure for determining the best reply to a behaviour strategy is applied to the probabilistic opponent model, then this leads to the a search method that we call *Probabilistic Opponent-Model Search* (PrOM search) (Donkers *et al.*, 2001).

### 6.2.1 Basic formulation

The first task is to transform the mixed strategy of the probabilistic opponent model into an equivalent behaviour strategy, that is, to determine the move probabilities $\Pr(m_j|h)$.[2] These probabilities are computed as follows. At a min node $h$, each opponent type $\omega_i$ plays Minimax and hence will select a move $m_j \in m(h)$ with a minimal value for that type $\omega_i$. If more than one move have the same minimal value for $\omega_i$, then either the first one is selected or one of these moves is selected at random. The probability that the opponent actually selects a certain move $m_j$ depends on the probabilities $\Pr(\omega_i)$ of the opponent types and the probability that a move is selected by an opponent type:

$$\Pr(m_j|h) = \textstyle\sum_i \Pr(\omega_i)\Pr(m_j|\,\omega_i, h) \tag{6.2}$$

The probability $\Pr(m_j|\,\omega_i, h)$ is either 0 or 1. It is 0 if the move is not minimal. It is 1 if the move is the first minimal move and the opponent type selects the first of the minimal moves otherwise it is 0. If the move is minimal and the opponent types select randomly between equipotent moves, the probability will be $1/K$ instead of 1, where $K$ is the number of equipotent moves $m_k$ with minimal value of $v_{\omega_i}(h+m_k)$. The values for $v_{\omega_i}(h)$ are computed by Minimax:

$$v_{\omega_i}(h) = \begin{cases} \max\limits_{m_j \in m(h)} \; v_{\omega_i}(h+m_j) & p(h) = \text{MAX} \\ \min\limits_{m_j \in m(h)} \; v_{\omega_i}(h+m_j) & p(h) = \text{MIN} \\ V_i(h) & h \in E \end{cases} \tag{6.3}$$

The PrOM-search value $v_0(h)$ of min node $h$ is the *expected* value of the child nodes' values $v_0(h+m_j)$:

$$v_0(h) = \textstyle\sum_j \Pr(m_j|h)\, v_0(h+m_j) \tag{6.4}$$

[2]In contrast with the Heuristic Program in Jansen (1992a) these probabilities are computed and not provided as part of the opponent model.

At max nodes, PrOM search just maximizes over the child node values and at leaf nodes the evaluation function $V_0$ is used directly. So the complete PrOM-search value is given by:

$$v_0(h) = \begin{cases} \max\limits_{m_j \in m(h)} \; v_0(h + m_j) & p(h) = \text{MAX} \\ \sum_j \Pr(m_j|h)\, v_0(h + m_j) & p(h) = \text{MIN} \\ V_0(h) & h \in E \end{cases} \tag{6.5}$$

### 6.2.2 An example

In figure 6.1, an example search tree is presented that demonstrates the working of PrOM search. The squares denote max nodes and the circles min nodes. The values inside the nodes are standard minimax values. In the example, there are two opponent types: $\omega_0$ and $\omega_1$ with probability 0.3 and 0.7, respectively. To the right of the nodes, the minimax values for these opponent types are given. By definition, the values for $\omega_0$ are equal to the minimax values. The PrOM-search values are above the nodes. The PrOM-search value of the left-hand min node is trivial to compute because both opponent types select the same move. The right-hand min node is more interesting, because $\omega_0$ would select the right-hand child ($v_{\omega_0} = 6$), but $\omega_1$ would select the left-hand child ($v_{\omega_1} = 8$). The expected value of this min node is equal to the weighed values of the two children (7.4), which is higher than the minimax value of this node (6), and even higher than the minimax value of the left-hand min node (7). At the root, PrOM search will thus select another move than Minimax does, which is indicated by the dotted line in the figure.

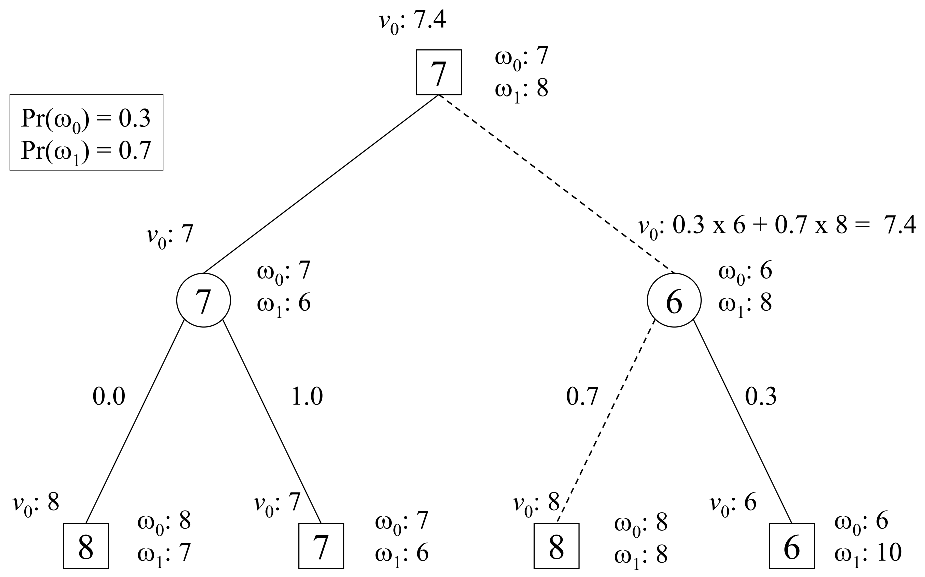


Figure 6.1: PrOM search and Minimax compared.

### 6.2.3 Related search methods

In section 2.1 we discussed five different search methods that preceded OM search. Among these five methods, both the *-Minimax method (Reibman and Ballard, 1983) and the heuristic program of Jansen (1992a; 1993) use move probabilities, like PrOM search. In Reibman and Ballard's approach, the move probabilities are assigned using a fixed formula based on the fallibility of the opponent and the ranking of the moves. In Jansen's approach the move probabilities are assigned on the basis of some heuristics concerning the positions reached by the moves. Both approaches constitute a behaviour strategy for MIN, not a mixed strategy like PrOM search.

There is also some similarity between PrOM search and $M^*$ search (see subsection 2.4.4). Both PrOM search and $M^*$ search use multiple opponent models. In $M^*$, the opponent is assumed to have a model of the player that can recursively include a second-level opponent model and so on. The way in which multiple opponent models are used in $M^*$ search is orthogonal to PrOM search: the approach of Carmel and Markovitch (1993) could be combined with PrOM search by taking $n$-level players instead of 1-level players for the $m$ opponent types in PrOM search. This would mean that at the leaf nodes $n \times m$ different evaluation functions have to be computed.

### 6.2.4 PrOM-search value

The PrOM-search value has a distinct relation with the minimax value and with OM-search values of a search tree. The following two lemmata are important and obvious.

**Lemma 1.** *If the probability of opponent type $\omega_0$ is 1, then PrOM search is equivalent to Minimax, and the PrOM-search value will be equal to the Minimax value.*

**Lemma 2.** *When the probability of another opponent type $\omega_j$ $(j > 0)$ is 1, then PrOM search is equivalent to OM search with opponent $\omega_j$, and the PrOM-search value will be equal to the OM-search value of opponent type $\omega_j$.*

The next property of PrOM search is less obvious:

**Theorem 3.** *The PrOM-search value $v_0$ is never less than the Minimax value $v_{mm}$ of that tree.*

*Proof.* Any search method $y$ that maximizes on max nodes, but backs-up an arbitrary *affine* combination of child-node values at min nodes, produces a search-tree value that is never less than the Minimax value. Namely, let $y$ be defined by:

$$v_y(h) = \begin{cases} \max\limits_{m_j \in m(h)} v_y(h+m_j) & p(h) = \text{MAX} \\ \sum_j \alpha_j v_y(h+m_j) \quad (\sum_j \alpha_j = 1,\ \alpha_j \geq 0) & p(h) = \text{MIN} \\ V(h) & h \in E \end{cases} \tag{6.6}$$

then $v_y(h) \geq v_{mm}(h)$ for all histories $h \in H$. At terminal histories $h \in E$ this is true, because $v_y(h) = v_{mm}(h) = V(h)$. Now assume that the inequality holds for all histories $h + m_j$. If $h$ is a max node, then it holds for $h$ because the maximum over $v_y(h + m_j)$ is necessarily equal to or greater than the maximum over $v_{mm}(h + m_j)$. If $h$ is a min node, then the inequality also holds, because:

$$\begin{aligned} & \forall_{m_j \in m(h)} : v_y(h+m_j) \geq v_{mm}(h+m_j) \geq v_{mm}(h) \\ \Rightarrow \quad & \textstyle\sum_j \alpha_j v_y(h+m_j) \geq v_{mm}(h) \\ \Rightarrow \quad & v_y(h) \geq v_{mm}(h) \end{aligned} \tag{6.7}$$

PrOM search is an instance of search method $y$ because the value at min nodes is computed by equation 6.4 where the probabilities form an affine combination of the child-node values (take $\alpha_j = \Pr(P_j)$). This means that the PrOM-search value is never less than the Minimax value. $\square$

Because OM-search is also an instance of algorithm $y$ (take $\alpha_j = 1$ if $v_{op}(P_j)$ is minimal, else take $\alpha_j = 0$), the proof of theorem 3 is an alternative proof for the first characteristic of OM search in subsection 2.3.3, namely, that the OM-search value is never less than the Minimax value.

Theorem 3 describes how the PrOM-search value of a given game tree behaves as a function of the opponent-type probabilities. In all remote corners of the probability space where one of the opponent types has probability 1, the PrOM-search value is either equal to the Minimax value (if $\Pr(\omega_0) = 1$), or equal to an OM-search value, which usually is higher than Minimax. Everywhere else, PrOM search returns a search-tree value that is at least as high as the Minimax value.

The exact form of the search-value function is difficult to predict. The determination of the PrOM-search value for a given opponent-type probability distribution is equivalent to the task of finding the maximum over a set of polynomials.[3] Every solution tree for MAX in the game tree represents a separate polynomial over the opponent-type probabilities. It is a polynomial function because of the repeated application of equation 6.4. Determining the PrOM-search value for a given probability distribution is the same as finding the strategy with the highest polynomial value for that distribution.

The search-value function is necessarily continuous and piece-wise smooth in the probability space, but it is not convex *per se*. It can have local minima and local maxima. The minima are always equal to or greater than the Minimax value, but the maxima can be higher than any of the OM-search values. In chapter 7 we present an experiment that proves that in PrOM search values can occur that are higher than the maximum OM-search value for each opponent type separately.

The search-value function being not convex makes it difficult to predict or approximate the PrOM-search value for a given probability distribution on the basis of the Minimax value and the OM-search values alone.

### 6.2.5 Risk

Using PrOM search implies three types of risks. First, the PrOM-search value is an *expected* value and not a guaranteed value. When MAX selects the most promising

[3]This is part of the mathematical field of nonlinear programming.

move having the highest expected value, MIN could by chance select a move that has a low probability but is inconvenient for MAX. This inherent risk of PrOM search could be adjusted to the risk-acceptance level of MAX by using a utility function $U(V(h))$ in formula 6.4 instead of the static evaluation function $V(h)$ to model either risk-seeking or risk-avoiding behaviour. Of course, the PrOM-search value would lose its relation with the minimax value of the tree and with the OM-search values when $U(V(h))$ is used instead of $V(h)$. In this thesis, we only investigate the risk-neutral form $V(h)$.

Second, the approximation of the opponent by the opponent types might deviate (considerably) from reality, causing the opponent to behave not as expected. Either the evaluation functions could be badly chosen, or the opponent-type probabilities could be wrong. The first error is inherent to OM search too, see subsection 2.3.4. The latter error is specific to PrOM search. In section 6.6 we will study the adaptation of these probabilities in case of observed prediction errors.

Third, especially when the probability on the first opponent type ($\omega_0$) is low, the same risk of running into type-I errors as discussed in section 4.2 is present in PrOM search. However, our hypothesis is that this risk is diminished by using the probabilistic opponent model. Experiments that test this hypothesis will be described in chapter 7.

## 6.3 Implementing PrOM Search

Since the PrOM-search method is derived from OM search, the implementation of PrOM search is similar to the OM-search implementations, and similar issues have to be dealt with. In subsection 6.3.1 we introduce two basic implementations. In subsection 6.3.2 we discuss how to deal with differing search depths and with restricted speculation is discussed. In subsection 6.3.3 we deal with pruning in PrOM search and in subsections 6.3.4 and 6.3.5 we add pruning to the two basic implementations of subsection 6.3.1. We briefly discuss bounded-sum pruning in subsection 6.3.6.

### 6.3.1 Basic implementations

Analogously to OM search, PrOM search can be implemented in a one-pass version and in a version with $\alpha$-$\beta$ probing. In this section, we provide implementations for both versions. However, we only treat the case in which the opponent types select just one move each at every min node and not the case in which the opponent types select multiple, equipotent moves.

The one-pass version is presented in algorithm 6.1. In this version, the node values for MAX ($v_0$) and for all opponent types ($v_{\omega_i}$) are computed in one pass. For this computation, the algorithm returns a vector containing all values for the opponent types, next to $v_0$ and the best move for MAX. At max nodes, the only task is to maximize all node values. At min nodes, the values for the opponent types have to be minimized, but also the move probabilities have to be computed and the expected value of $v_0$ has to be computed. The vector $\bar{w}^*$ is used to store the value $v_0(h + m_j)$ in case node $h + m_j$ had the lowest value for opponent type $\omega_i$ so

```
PromSearch1p($h$)
1  if ($h \in E$) return ($V_0(h), [V_0(h), V_1(h), \ldots, V_{n-1}(h)]$, null)
2  $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$) ; $m_0^* \leftarrow m$
3  if ($p(h) =$ MAX)
4      $v_0^* \leftarrow -\infty$ ; $\bar{v}^* \leftarrow [-\infty, \ldots, -\infty]$
5      while ($m \neq$ null)
6          $(v_0, \bar{v}, mm) \leftarrow$ PromSearch1p($h + m$)
7          if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m_0^* \leftarrow m$
8          for $i \in \{0, \ldots, n-1\}$ if ($\bar{v}_i > \bar{v}_i^*$) $\bar{v}_i^* \leftarrow \bar{v}_i$
9          $m \leftarrow$ nextMove($L$)
10 if ($p(h) =$ MIN)
11     $\bar{v}^* \leftarrow [+\infty, \ldots, +\infty]$; $\bar{w}^* \leftarrow [0, \ldots, 0]$
12     while ($m \neq$ null)
13         $(v_0, \bar{v}, mm) \leftarrow$ PromSearch1p($h + m$)
14         for $i \in \{0, \ldots, n-1\}$ if ($\bar{v}_i < \bar{v}_i^*$) $\bar{v}_i^* \leftarrow \bar{v}_i$ ; $\bar{w}_i^* \leftarrow v_0$
15         $m \leftarrow$ nextMove($L$)
16     $v_0^* \leftarrow 0$ ; $m_0^* \leftarrow$ null
17     for $i \in \{0, \ldots, n-1\}$ $v_0^* \leftarrow v_0^* + \Pr(\omega_i)\, \bar{w}_i^*$
18 return $(v_0^*, \bar{v}^*, m_0^*)$
```

Algorithm 6.1: One-pass PrOM search.

far. In line 17, these values are multiplied by the opponent-type probabilities $\Pr(\omega_i)$ and summed to obtain the value of $v_0$. This approach is justified by the following formulae:

$$\begin{aligned} v_0(h) &= \sum_j \Pr(m_j|h)\, v_0(h+m_j) \\ &= \sum_j \sum_i \Pr(\omega_i) \Pr(m_j|\,\omega_i, h)\, v_0(h+m_j) \\ &= \sum_i \Pr(\omega_i)\, (\sum_j \Pr(m_j|\,\omega_i, h)\, v_0(h+m_j)) \end{aligned} \tag{6.8}$$

since $\Pr(m_j|\,\omega_i, h) = 1$ when node $m_j$ is selected by opponent type $\omega_i$. With some additional adminstration, the same algorithm can be used in the case that the opponent types are allowed to select multiple equipotent moves.

The version of PrOM search with $\alpha$-$\beta$ probes is given in algorithm 6.2. The node values for the opponent types are determined in standard calls to the $\alpha$-$\beta$ procedure, similar to OM search with $\alpha$-$\beta$ probes (algorithm 3.3). This means that these values do not have to be returned by this algorithm. At max nodes, only the maximum value for MAX has to determined. The handling of min nodes is more elaborate. In a first loop, a separate $\alpha$-$\beta$ probe is performed for each opponent type. The moves returned by the probes are collected in a set $L$ (without duplicates). In a second loop, for each move in $L$, the algorithm is called recursively and the resulting values are collected. The computation of the expected value for MAX is performed again on the basis of formula 6.8. The condition $(m = \bar{m}_i^*)$ in line 17 indicates a probability of 1 for the current move $m$ and opponent type $\omega_i$.

```
PromSearchPb($h$)
1   if ($h \in E$) return ($V_0(h)$, null)
2   if ($p(h) = $ MAX)
3       $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$) ; $m_0^* \leftarrow m$
4       $v_0^* \leftarrow -\infty$
5       while ($m \neq$ null)
6           $(v_0, mm) \leftarrow$ PromSearchPb($h + m$)
7           if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m_0^* \leftarrow m$
8           $m \leftarrow$ nextMove($L$)
9   if ($p(h) = $ MIN)
10      $L \leftarrow \varnothing$
11      for $i \in \{0, \ldots, n-1\}$
12          $(v, \bar{m}_i^*) \leftarrow$ AlphaBetaSearch($h, -\infty, \infty, V_i(\cdot)$)
13          $L \leftarrow L \cup \{\bar{m}_i^*\}$
14      $v_0^* \leftarrow 0$; $m_0^* \leftarrow$ null ; $m \leftarrow$ firstMove($L$)
15      while ($m \neq$ null)
16          $(v_0, mm) \leftarrow$ PromSearchPb($h + m$)
17          for $i \in \{0, \ldots, n-1\}$ if ($m = \bar{m}_i^*$) $v_0^* \leftarrow v_0^* + \Pr(\omega_i)\, v_0$
18          $m \leftarrow$ nextMove($L$)
19  return ($v_0^*, m_0^*$)
```

Algorithm 6.2: PrOM search with $\alpha$-$\beta$ probing.

In order to make this algorithm suitable for multiple selections by the opponent types, first the $\alpha$-$\beta$ probes must be changed so that they return a list of moves selected by the opponent type. A vector $\bar{n}$ can store the number of moves selected by each type. The lists have to be combined into a single list $L$ with no duplicates. Finally, the assignment in line 17 has to change to: $v_0^* \leftarrow v_0^* + \Pr(\omega_i)\, \bar{w}_i^* / \bar{n}_i$.

### 6.3.2 Differing search depths and restricted speculation

In subsections 3.1.3 and 3.1.4 we discussed two refinements of OM search that also are applicable to PrOM search: unequal search depths and restricted speculation. In PrOM search, the two refinements are preferably combined since the computation of move probabilities at min nodes requires the determination of the value for all opponent types. Different search depth per opponent type (different sets of terminal histories $E_{\omega_i}$) can be allowed as long as all opponent types include at least the set of speculative positions $H_{spec}$. Algorithm 6.3 shows an implementation of PrOM search with $\alpha$-$\beta$ probes and these refinements. The only visible change with respect to algorithm 6.2 is in line 1. The other change is within the $\alpha$-$\beta$ probes per opponent type. These probes should obey the opponent-type dependent sets $E_{\omega_i}$.

### 6.3.3 Pruning in PrOM search

The core of $\alpha$-$\beta$ search is the pruning of the search tree. The appropriateness of PrOM search will increase with the possibility to prune the search tree according to

```
PromSearchEnh($h$)
1   if ($h \notin H_{spec}$ or $h \in E_0$) return AlphaBetaSearch($h, -\infty, \infty, V_0(\cdot)$)
2   if ($p(h) = $ MAX)
3       $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$) ; $m_0^* \leftarrow m$
4       $v_0^* \leftarrow -\infty$
5       while ($m \neq$ null)
6           $(v_0, mm) \leftarrow$ PromSearchEnh($h + m$)
7           if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m_0^* \leftarrow m$
8           $m \leftarrow$ nextMove($L$)
9   if ($p(h) = $ MIN)
10      $L \leftarrow \varnothing$
11      for $i \in \{0, \ldots, n-1\}$
12          $(v, \bar{m}_i^*) \leftarrow$ AlphaBetaSearch($h, -\infty, \infty, V_i(\cdot)$)
13          $L \leftarrow L \cup \{\bar{m}_i^*\}$
14      $v_0^* \leftarrow 0$; $m_0^* \leftarrow$ null ; $m \leftarrow$ firstMove($L$)
15      while ($m \neq$ null)
16          $(v_0, mm) \leftarrow$ PromSearchEnh($h + m$)
17          for $i \in \{0, \ldots, n-1\}$ if ($m = \bar{m}_i^*$) $v_0^* \leftarrow v_0^* + \Pr(\omega_i)\, v_0$
18          $m \leftarrow$ nextMove($L$)
19  return $(v_0^*, m_0^*)$
```

Algorithm 6.3: PrOM search with $\alpha$-$\beta$ probing enhanced with differing search depths and restricted speculation.

the standards of $\alpha$-$\beta$ search. Since PrOM search is derived from OM search it seems plausible that the pruning techniques used in OM search can be applied to PrOM search too. Just as in OM search, a branch at a max node can be pruned as soon as it is clear that the max node will not be selected at the parent min node. In that case, none of the information in the subtree under that max node has any influence on the value of its predecessors including the root of the search tree. Pruning at a min node is not possible in PrOM search because the values of all child nodes (with non-zero move probability) of a min node are needed to decide whether that min node will be selected by its parent max node. It is not possible to determine an intermediate upper bound for the value of the min node that would lead to pruning.

In OM search, pruning at a max node can happen as soon as it becomes clear that the opponent will select another move at the parent min node. This pruning is controlled by a $\beta$ value that is passed from sibling to sibling at min nodes (and to their child nodes). Hence the pruning method in OM search was called $\beta$ pruning. In PrOM search, it is only certain that a max node $h + m$ will not be needed at a parent min node $h$ when the move probability $\Pr(m|h)$ is zero. This move probability is determined by the selection of all opponent types and the opponent-type probabilities. The probability of a move is zero when none of the opponent types with a non-zero opponent-type probability selects that move. If we assume that all opponent-type probabilities are larger than zero, then the remaining demand is that the max node must be selected by none of the opponent types. We will discuss pruning in PrOM search separately for the one-pass version (subsection 6.3.4) and the version with $\alpha$-$\beta$ probing (subsection 6.3.5).

### 6.3.4 Pruning in one-pass PrOM search

Whether a max node will be selected by any opponent type can be checked for all opponent types simultaneously in one-pass PrOM search by using a separate $\beta_i$ value *per* opponent type $\omega_i$. These $\beta_i$ values are the minimum values of $v_{\omega_i}$ so far among the sibling nodes. As soon as all $\beta_i$ values are exceeded by the corresponding $v_{\omega_i}$ values of (at least) one of the child nodes, the remaining subtree under the max node can be pruned. So, "$\beta$-pruning" is an appropriate name for pruning in one-pass PrOM search too.

In one-pass PrOM search, the $\beta_i$ values can safely be passed from a max node to the next sibling max node. This is called *shallow* $\beta$ pruning in $\alpha$-$\beta$ search. Figure 6.2 shows an example of shallow pruning. The values for $v_{\omega_0}$ and $v_{\omega_1}$ at node ***b*** are passed as $\beta_i$ values to node ***c***. After the evaluation at node ***f*** it is clear that both $\beta_i$ values are exceeded, which means that node ***c*** will not be selected by any opponent type at node ***a***. Further investigation of node ***c*** is meaningless.

Passing $\beta_i$ values to child nodes in order to perform *deep* $\beta$ pruning is impossible in one-pass PrOM search. Figure 6.3 shows an example of the problems with deep $\beta$ pruning. The values for $v_{\omega_0}$ and $v_{\omega_1}$ at node ***b*** are both the lowest values so far between the child nodes of node ***a*** so they act as $\beta_i$ values for node ***c***. In deep pruning, these $\beta_i$ values are passed to the nodes ***d*** and ***e***. At node ***g***, both values $v_{\omega_0}$ and $v_{\omega_1}$ exceed the $\beta_i$ values. It would seem reasonable to prune the rest of the subtree under node ***e***, but this is not true. The values at node ***f*** are such that for opponent type 1, node ***f*** would be preferred over node ***e***, so the value of $v_{\omega_1}$ at node

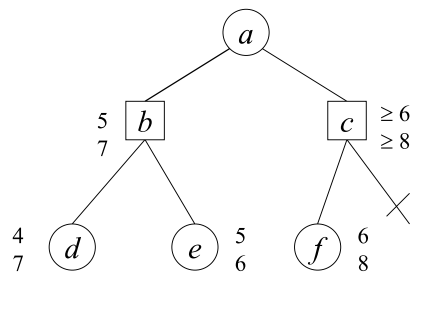


Figure 6.2: An example to illustrate shallow $\beta$-pruning in one-pass PrOM search. Only the values for $v_{\omega_0}$ and $v_{\omega_1}$ are given, next to the nodes.

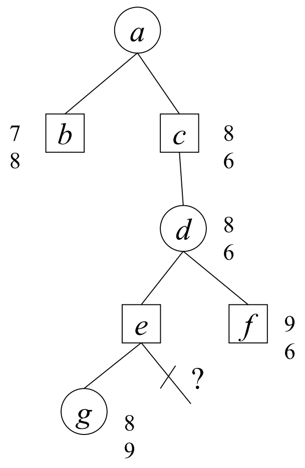


Figure 6.3: An example to illustrate why deep $\beta$-pruning in one-pass PrOM search is not possible. Only the values for $v_{\omega_0}$ and $v_{\omega_1}$ are given, next to the nodes. The question mark indicates a spot where pruning is disputable.

```
PromSearchBeta1p($h, \bar{\beta}$)
1  if ($h \in E$) return ($V_0(h), [V_0(h), V_1(h), \ldots, V_{n-1}(h)]$, null)
2  $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$) ; $m_0^* \leftarrow m$
3  if ($p(h) =$ MAX)
4      $v_0^* \leftarrow -\infty$ ; $\bar{v}^* \leftarrow [-\infty, \ldots, -\infty]$
5      while ($m \neq$ null)
6          ($v_0, \bar{v}, mm$) $\leftarrow$ PromSearchBeta1p($h + m, \bar{\beta}$)
7          if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m_0^* \leftarrow m$
8          for $i \in \{0, \ldots, n-1\}$ if ($\bar{v}_i > \bar{v}_i^*$) $\bar{v}_i^* \leftarrow \bar{v}_i$
9          $prune \leftarrow$ true ; for $i \in \{0, \ldots, n-1\}$ if ($\bar{v}_i^* < \beta_i$) $prune \leftarrow$ false
10         if ($prune$) $m \leftarrow$ null else $m \leftarrow$ nextMove($L$)
11 if ($p(h) =$ MIN)
12     $\bar{v}^* \leftarrow [+\infty, \ldots, +\infty]$; $\bar{w}^* \leftarrow [0, \ldots, 0]$
13     while ($m \neq$ null)
14         ($v_0, \bar{v}, mm$) $\leftarrow$ PromSearchBeta1p($h + m, \bar{v}^*$)
15         for $i \in \{0, \ldots, n-1\}$ if ($\bar{v}_i < \bar{v}_i^*$) $\bar{v}_i^* \leftarrow \bar{v}_i$ ; $\bar{w}_i^* \leftarrow v_0$
16         $m \leftarrow$ nextMove($L$)
17     $v_0^* \leftarrow 0$ ; $m_0^* \leftarrow$ null
18     for $i \in \{0, \ldots, n-1\}$ $v_0^* \leftarrow v_0^* + \Pr(\omega_i)\, \bar{w}_i^*$
19 return ($v_0^*, \bar{v}^*, m_0^*$)
```

Algorithm 6.4: One-pass $\beta$-pruning PrOM search.

***d*** is smaller than $\beta_1$. At the same time, opponent type 0 will prefer node ***e*** over node ***f***. The values at node ***c*** indicate that the move probabilities for both nodes ***b*** and ***c*** are non-zero at node ***a***, so the information of node ***e*** is needed to compute the value of node ***a***. This means that pruning at node ***e*** is prohibited.

Algorithm 6.4 shows the implementation of $\beta$-pruning in one-pass PrOM search. The check whether pruning is allowed is performed in line 9 and the actual pruning takes place in line 10. The $\beta$ parameter in the recursive call in line 6 is set to $\bar{\beta}$, but since deep pruning is not possible, this value is ignored at min nodes. Below, we will denote this algorithm by PrOM$^{\beta 1p}$.

### 6.3.5 Pruning in PrOM search with $\alpha$-$\beta$ probing

The shallow $\beta$ pruning as described above is intrinsic to PrOM search with $\alpha$-$\beta$ probes as described in algorithm 6.2. At a min node, every separate $\alpha$-$\beta$ probe will stop executing the evaluation of a child max node as soon as it is clear that this max node does not influence the value of the parent min node. Any part of the subtree that is pruned in $\beta$-pruning one-pass PrOM search is also not investigated by PrOM search with $\alpha$-$\beta$ probing since all individual $\alpha$-$\beta$ probes will ignore (at least) these parts of the tree.

Nevertheless, more pruning can be added to algorithm 6.2. Similar to $\beta$ pruning in OM search with $\alpha$-$\beta$ probes (see subsection 3.2.3), it is not necessary to call always the $\alpha$-$\beta$ probes with open $[-\infty, +\infty]$ windows, but the decision whether to close a windows is more delicate. Figure 6.4 (originally figure 3.2, repeated here for the

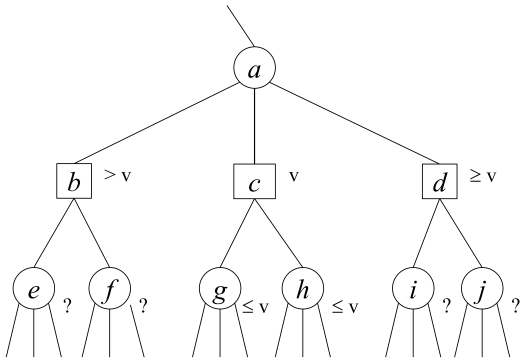


Figure 6.4: An example to illustrate the $\beta$ parameter of $\alpha$-$\beta$ probes.

readers' convenience) can be used to illustrate how the $\beta$ parameters for the probes in PrOM search can be set. Assume that an opponent type has selected the move leading to ***c***. Since no bound can be given for the subgame values of nodes ***e***, ***f***, ***i***, and ***j***, $\alpha$-$\beta$ probes for this opponent type must have $[-\infty, +\infty]$ windows. The bound on the values for nodes ***g*** and ***h*** allows a window of $[-\infty, v+1]$ for the $\alpha$-$\beta$ probes at these nodes.

```
PromSearchBetaPb($h, \bar{\beta}$)
1   if ($h \in E$) return ($V_0(h)$, null)
2   if $p(h)$ = MAX
3       $L \leftarrow m(h)$ ; $m \leftarrow$ firstMove($L$) ; $m_0^* \leftarrow m$
4       $v_0^* \leftarrow -\infty$
5       while ($m \neq$ null)
6           $(v_0, mm) \leftarrow$ PromSearchBetaPb($h+m, \bar{\beta}$)
7           if ($v_0 > v_0^*$) $v_0^* \leftarrow v_0$ ; $m_0^* \leftarrow m$
8           $m \leftarrow$ nextMove($L$)
9   if $p(h)$ = MIN
10      $L \leftarrow \varnothing$
11      for $i \in \{0, \ldots, n-1\}$
12          $(\bar{v}_i^*, \bar{m}_i^*) \leftarrow$ AlphaBetaSearch($h, -\infty, \bar{\beta}_i, V_i(\cdot)$)
13          $L \leftarrow L \cup \{\bar{m}_i^*\}$
14      $v_0^* \leftarrow 0$; $m_0^* \leftarrow$ null ; $m \leftarrow$ firstMove($L$)
15      while ($m \neq$ null)
16          for $i \in \{0, \ldots, n-1\}$ if ($m = \bar{m}_i^*$) $\bar{\beta}_i \leftarrow \bar{v}_i^* + 1$ else $\bar{\beta}_i \leftarrow \infty$
17          $(v_0, mm) \leftarrow$ PromSearchBetaPb($h+m, \bar{\beta}$)
18          for $i \in \{0, \ldots, n-1\}$ if ($m = \bar{m}_i^*$) $v_0^* \leftarrow v_0^* + \Pr(\omega_i)\, v_0$
19          $m \leftarrow$ nextMove($L$)
20  return ($v_0^*, m_0^*$)
```

Algorithm 6.5: $\beta$-pruning PrOM search with $\alpha$-$\beta$ probing.

To summarize, the $\beta$ parameter for the $\alpha$-$\beta$ probe of an opponent type at a min node can be set to $v{+}1$ when the parent max node was selected by that opponent type and the value of the parent max node is $v$. Algorithm 6.5 shows the implementation of this pruning in PrOM search with $\alpha$-$\beta$ probing[4]. We will denote this algorithm by the abbreviation PrOM$^{\beta Pb}$.

### 6.3.6 Bounded-sum pruning

A pruning method similar to the method developed by Carmel and Markovitch (1996a) for M* search, which we called bounded-sum pruning in subsection 3.2.4, can be applied to PrOM search too. The condition on the opponent types should be restated for bounded pruning as follows: for all histories in $E$ and for all opponent types $\omega_i$ $(i > 0)$, the distance between $V_0(h)$ and $V_i(h)$ should not exceed a given bound $B$. If this is the case, it is possible to allow limited deep pruning in PrOM$^{\beta 1p}$. The condition on the evaluation of leaf nodes implies that the values of $v_{\omega_0}$ and $v_{\omega_i}$'s on internal nodes are also bound to a difference $B$. So if, in figure 6.3, the value for $v_{\omega_1}$ of node ***c*** is known to exceed the current value of $\bar{\beta}_1$, something can be said on the value of $v_{\omega_0}$ for that node too, since $v_{\omega_0}$ will not be smaller than $v_{\omega_1} - B$. If the extent to which $v_{\omega_1}$ exceeds the current $\bar{\beta}_1$ is sufficiently large, it can be deduced that node ***c*** will also not be selected by opponent type $\omega_0$ at node **a**. Pruning at node ***e*** would then be safe.

In PrOM$^{\beta Pb}$, bounded-sum pruning can be used in the form of adjusting the $\alpha$ and $\beta$ parameters in the subsequent $\alpha$-$\beta$ probes at a min node. The value returned from the first probe minus $B$ can, for instance, be used as lower bound for the other probes.

## 6.4 Search Enhancements

In this section we briefly discuss the same search enhancements as discussed in section 3.4. Most of the search enhancements applicable to OM search are also of use in PrOM search, but there are some differences. Furthermore, there is one additional enhancement that is specific to PrOM search. We start with this enhancement.

### 6.4.1 Shared evaluation

In PrOM search an opponent type is represented by the evaluation function $V_{\omega_i}$, but nothing is stated on the nature of this evaluation function. In some cases, the evaluation of a position is a weighted sum over a series of numbers, each number being the result of a costly computation for a distinct evaluation component. The difference between opponent types could be caused by the way in which these costly computations are performed, but more likely, the difference between opponent types stems simply from differences in weights that are assigned to the components of the evaluation. In the latter case, the opponent types can share a large amount of the computations needed for their evaluation function.

[4]In Donkers *et al.* (2001), the same algorithm is called $\beta$-passing PrOM search.

If the opponent types can share the costly part of their evaluation function, the efficiency of PrOM search will improve greatly. In the one-pass implementations of PrOM search, it is trivial to apply shared evaluation since all evaluation functions are applied at the same time on a leaf node (e.g., line 1 in algorithm 6.1). But in the probing implementations, shared evaluation can be applied too. When for the first time the position of a leaf node is evaluated for one opponent type, the results of the distinct evaluation components can be stored (for instance, in a transposition table) and reused for other opponent types that need to evaluate the position at that leaf node.

The same holds for OM search: when at a leaf node $h$ the value $V_{op}(h)$ has been computed, the result of the shared part of the evaluation can be reused to compute $V_0(h)$. However, the gain of using shared computation will be less in OM search than in PrOM search because OM search deals with two evaluation functions and PrOM search possibly with many. Moreover, pruning happens more in OM search than in PrOM search. This means that the number of leaf nodes for which $V_{op}(h)$ and $V_0(h)$ have to be performed both is relatively low.

Carmel and Markovitch (see 1998, page 332) mention a similar efficiency improvement in $\mathrm{M}^*_{1\text{-}pass}$ where multiple evaluations take place at leaf nodes.

### 6.4.2 Move ordering

Move ordering in PrOM search is more delicate than in OM search since the move ordering should be profitable for all opponent types together. Inside $\alpha$-$\beta$ probes for a given opponent type, move ordering can be tuned to that opponent type, but in $\mathrm{PrOM}^{\beta 1p}$ and in the max nodes of $\mathrm{PrOM}^{\beta Pb}$, the availability of a good ordering depends on the correlation between the opponent types. The higher the correlation, the better ordering is possible. History tables and the killer-move heuristic are applicable inside $\alpha$-$\beta$ probes, but separate tables are needed per opponent type.

### 6.4.3 Transposition tables and iterative deepening

The use of transposition tables is evidently important inside the $\alpha$-$\beta$ probes of $\mathrm{PrOM}^{\beta Pb}$. In chapter 7 we will conduct experiments to measure the impact of transposition tables on the efficiency of $\mathrm{PrOM}^{\beta Pb}$. A transposition table can also be used to store the value for MAX, but similar to OM search, the use of this table is limited.

When shared evaluation is applicable, instead of using a separate transposition table per opponent type, one common transposition table can be used for all opponent types together. In this way, $\mathrm{PrOM}^{\beta Pb}$ is capable of profiting from shared evaluation (see subsection 6.4.1).

### 6.4.4 Search extensions and forward pruning

The techniques for search extensions and forward pruning are applicable inside the $\alpha$-$\beta$ probes of $\mathrm{PrOM}^{\beta Pb}$, but the same care should be taken with these enhancements as in OM search. In $\mathrm{PrOM}^{\beta 1p}$, it is only possible to apply these techniques for MAX and for all opponent types simultaneously. The computation of move probabilities might however be an issue when not all moves at a position are evaluated. It is not clear how move probabilities should be assigned in a node that is investigated by one of the opponent types but not by at least one of the other types: should all opponent types be involved or should only the opponent types be involved that investigate the node? This issue is analogous to the counterfactual problems that we discussed in subsection 2.5.2.

### 6.4.5 Endgame databases

Similar to OM search, the availability of an endgame database can be taken as part of an opponent type. For instance, when it is not completely certain to which endgame databases the opponent has access to (e.g., the three-men's or four-men's databases in chess), it could be useful to design an opponent model of different types, all sharing the same evaluation function, but each one having access to different endgame databases.

### 6.4.6 Aspiration search

The application of PV search (see subsection 3.4.6) inside $\alpha$-$\beta$ probes of PrOM$^{\beta Pb}$ is just as valid and profitable as in OM search. Of course, this enhancement should be combined with transposition tables and iterative deepening. Anywhere else, there is not much room for aspiration search in PrOM search.

## 6.5 Best-case Analyses

The best-case analysis for $\beta$-pruning PrOM search is analogous to the best-case analysis for OM search in section 3.3. We will first define the best case for pruning in PrOM search. Then we will deduce the best-case complexity for the one-pass version (subsection 6.5.1) and the version with $\alpha$-$\beta$ probing (subsection 6.5.2). Finally we will discuss the results (subsection 6.5.3).

The best case for pruning in PrOM search is defined as a game tree in which the children $h + m_j$ of every node are ordered *simultaneously* on all values $v_{\omega_j}(h + m_j)$, for max nodes in decreasing order and for min nodes in increasing order. It is clear that such node ordering does not exist in all game trees. The best-case analysis that is presented here assumes a tree in which such a simultaneous ordering for all opponent types is possible. Furthermore, we assume that the game tree is uniform with depth $d$ and branching factor $w$, and that there are $n$ opponent types.

### 6.5.1 Best-case analysis of PrOM$^{\beta 1p}$

Figure 6.5 gives a schematic representation of the best-case game tree for one-pass $\beta$-pruning PrOM search. The difference with the similar game tree of figure 3.4 is the absence of deep pruning: the $\bar{\beta}$ values are not passed to the children of max nodes. This means that the subtree at the leftmost grandchild of each B-node is congruent to a subtree at an A-node since no pruning can take place directly under

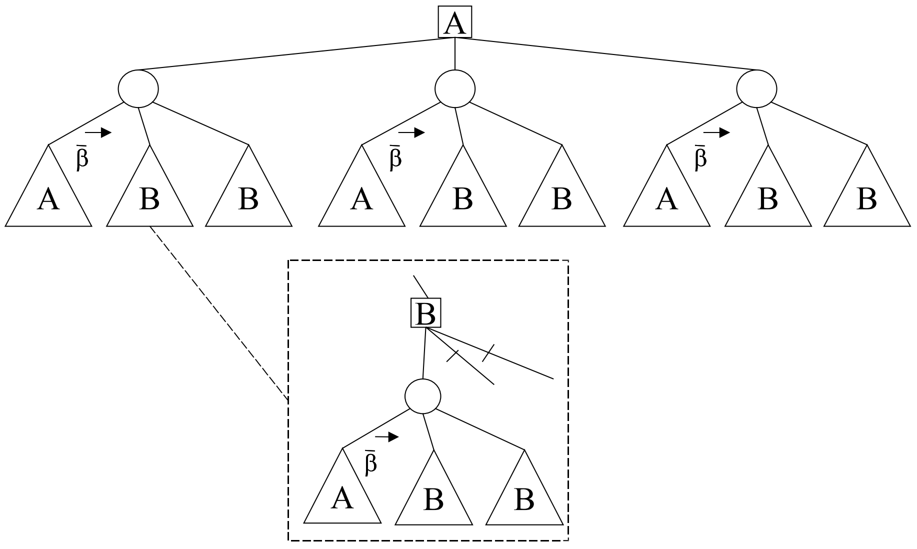


Figure 6.5: Theoretically best-case tree for one-pass $\beta$-pruning OM search.

this grandchild node. The number of evaluations needed in this tree for one-pass $\beta$-pruning PrOM search is given by the following formulae:

$$\begin{aligned} C_{PrOM^{\beta_{1p}}}(d,w,n) &= (2w-1)\,C_{PrOM^{\beta_{1p}}}(d-2,w,n) \\ C_{PrOM^{\beta_{1p}}}(1,w,n) &= n\,w \\ C_{PrOM^{\beta_{1p}}}(2,w,n) &= n\,w^2 \end{aligned} \tag{6.9}$$

These formulae can be understood as follows. Figure 6.5 shows that the number of evaluations needed under A-nodes is $w$ times the number of evaluations needed under B-nodes of the same depth. For an A-node of depth $d$, $w$ times an A-node of depth $d-2$ is needed plus $w(w-1)$ times a B-node of depth $d-2$. The $w(w-1)$ B-nodes together take the same amount of evaluations as $w-1$ A-nodes of depth $d-2$. In total, this leads to $(2w-1)$ times an A-node of depth $d-2$. For nodes of depth 1, $n\,w$ evaluations are needed, one for every opponent type at every child node. Nodes of depth 2 take $n\,w^2$ evaluations, since no pruning can occur. The three formulae (6.9) are easily put in a single closed form:

$$C_{PrOM^{\beta_{1p}}}(d,w,n) = n\,(2w-1)^{\lceil d/2\rceil-1}\,w^{2+\lfloor d/2\rfloor-\lceil d/2\rceil} \tag{6.10}$$

This closed form can be found by repeated substitution; its validity can be checked by complete induction.

## 6.5.2 Best-case analysis of PrOM$^{\beta Pb}$

The best-case of $\beta$-pruning PrOM search with $\alpha$-$\beta$ probes, can be derived from the analysis of OM$^{\beta Pb}$ in subsection 3.3.2. An important observation is that in the best case, all opponent types will select the same move at every min node (i.e., the first move), so only one $\alpha$-$\beta$ probe is performed at every min node per opponent type. A further observation is that at leaf nodes, no separate evaluation for MAX is needed, since the evaluation for opponent type $\omega_0$ can be used.[5] This means that all evaluations are performed inside the probes. The number of evaluations in every probe of depth $d$ is, in the best case, equal to $C_{\alpha-\beta}(d,w)$. The number of evaluations needed in PrOM$^{\beta Pb}$ thus is:

$$\begin{aligned} C_{PrOM^{\beta Pb}}(d,w,n) &= n \sum_{i=1}^{\lceil d/2 \rceil} w^i \, C_{\alpha-\beta}(d-2i+1,w) \\ &= n \sum_{i=1}^{\lceil d/2 \rceil} w^i \, (w^{\lfloor (d-2i+1)/2 \rfloor} + w^{\lceil (d-2i+1)/2 \rceil} - 1) \\ &= n \sum_{i=1}^{\lceil d/2 \rceil} w^i \, w^{\lfloor (d-2i+1)/2 \rfloor} + w^i \, w^{\lceil (d-2i+1)/2 \rceil} - w^i \\ &= n \, \lceil d/2 \rceil (w^{\lfloor (d+1)/2 \rfloor} + w^{\lceil (d+1)/2 \rceil}) - n \, \frac{w^{\lceil d/2 \rceil + 1} - w}{w-1} \end{aligned} \tag{6.11}$$

## 6.5.3 A comparison

The following two formulae summarize the best-case analysis of $\beta$-pruning PrOM search:

| | |
|---|---|
| PrOM$^{\beta 1p}$: | $n \, (2w-1)^{\lceil d/2 \rceil - 1} \, w^{2 + \lfloor d/2 \rfloor - \lceil d/2 \rceil}$ |
| PrOM$^{\beta Pb}$: | $n \, \lceil d/2 \rceil (w^{\lfloor (d+1)/2 \rfloor} + w^{\lceil (d+1)/2 \rceil}) - n \, \frac{w^{\lceil d/2 \rceil + 1} - w}{w-1}$ |

While these formulae might not immediately reveal a preference, figure 6.6 shows clearly that on the basis of the best-case analysis, the version of $\beta$ pruning with $\alpha$-$\beta$ probes is to be preferred. The absence of deep-pruning in the one-pass version causes a significant worse behaviour in terms of the number of evaluations per $\alpha$-$\beta$ evaluation.

Even in the case that shared evaluation is possible, its advantage for one-pass version is not enough since the best-case time complexity is linear in the number of opponent types. Moreover, a careful application of transposition tables in the $\alpha$-$\beta$ probes also allows for reuse of computations.

[5]However, care has to be taken in case of an asymmetric evaluation function that evaluates a position differently for MAX and for MIN.

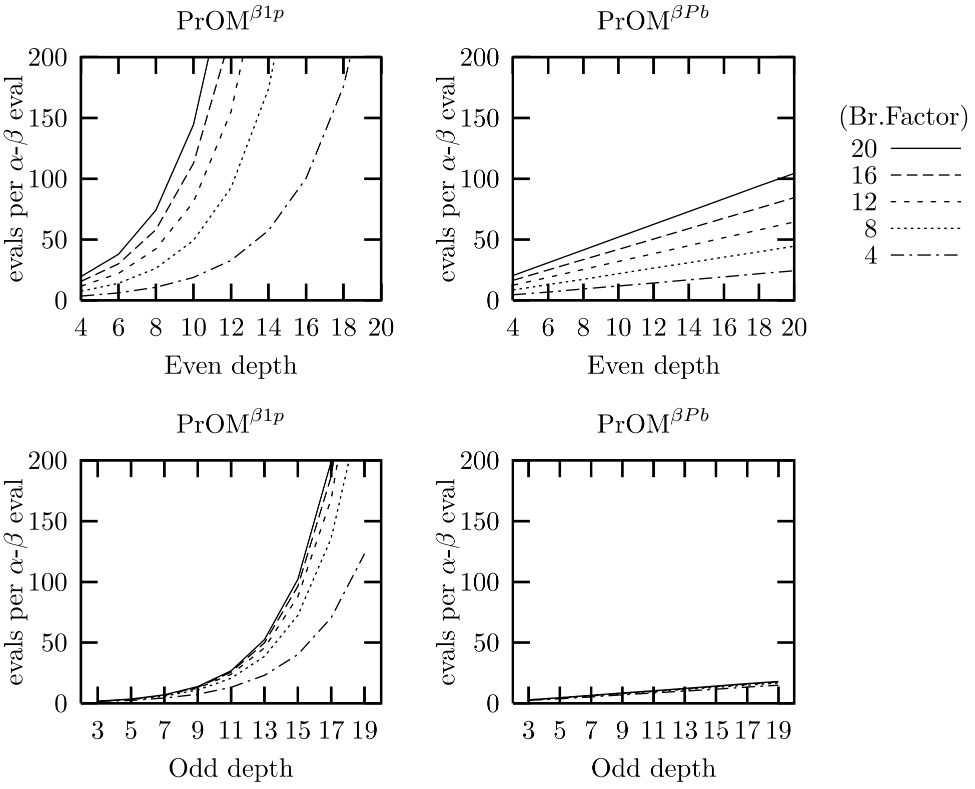


Figure 6.6: Best-case results of PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$ compared. The graphs show the number of evaluations *per* $\alpha$-$\beta$ evaluation *per* opponent type.

## 6.6 Learning a Probabilistic Opponent Model

The probabilistic opponent model of PrOM search consists of two parts: the set of opponent types and the opponent-type probabilities. These parts need their own approach in learning. In this thesis, we concentrate on learning of the opponent-type probabilities for a given set of opponent types and we will only briefly discuss learning of the opponent types themselves.

The task that we want to solve concerning opponent-type probabilities is: given a set of opponent types $\omega_0 \ldots \omega_n$, what is the best probability distribution over these types to be used by PrOM search against a given target opponent $\Omega$? This learning task has two manifestations: *off-line* learning and *on-line* learning. In off-line learning, we have available a (preferably large) set of positions together with the moves that our target opponent $\Omega$ selects. During on-line learning, we only have the few moves by $\Omega$ that are observed during the actual play.

### 6.6.1 Off-line learning of probabilities

Let $T$ be a recorded (randomized) set of positions, $T \subseteq H$ and let the function $m_\Omega : T \rightarrow M$ indicate the moves selected by the target opponent $\Omega$ in these positions. Let also be given an opponent model $\Omega(\bar{\pi})$ which consists of a set of opponent types $\omega_0, \ldots, \omega_n$ and a probability distribution $\bar{\pi}$, where $\bar{\pi}_i = \Pr(\omega_i)$. The objective of off-line learning is to find, using $m_\Omega(\cdot)$ and $T$, a probability distribution $\bar{\pi}$ such that PrOM search with $\Omega(\bar{\pi})$ plays best against $\Omega$.

The definition of PrOM search suggests that this probability distribution is the same as the probability distribution that *predicts* the behaviour of $\Omega$ the best. This hypothesis seems straightforward, but only actual game-playing can prove the correctness. We define the best predictor as the probability distribution $\bar{\pi}$ that maximizes the likelihood of $\Omega(\bar{\pi})$ selecting the same moves as $\Omega$ at the positions in $T$. In off-line learning, we do not prefer one probability distribution over another one, so the maximum likelihood estimator (MLE) is the appropriate method to obtain $\bar{\pi}$ (see Mood, Graybill, and Boes, 1974, page 276).

Before we can determine the MLE, a closer look at the set of positions is needed in relation to the opponent types. Define a subset $H_{\omega_i} \subseteq H$ for every opponent type $\omega_i$ as the set of positions $h$ such that the move $m_{\omega_i}(h)$ selected by $\omega_i$ at $h$ is not selected by any of the other opponent types. $H_{\omega_i}$ contains all positions that are said to be *unambiguous* with respect to $\omega_i$. Define further $\Pr(h_{\omega_i})$ as the probability of a position to be unambiguous with respect to $\omega_i$. The set $H$ is finite, since we study only finite games, so $\Pr(h_{\omega_i})$ is equal to $\# H_{\omega_i} / \# H$. We assume that $\Pr(h_{\omega_i}) > 0$ for all opponent types $\omega_i$. (If $\Pr(h_{\omega_i}) = 0$, the opponent type can be neglected.)

The likelihood that $\Omega(\bar{\pi})$ selects move $m_\Omega(h)$ and that the selection is unambiguous for the opponent type that selected the move, is given by:

$$\mathrm{L}(m_\Omega(h) \,|\, \bar{\pi}) = \bar{\pi}_i \Pr(h_{\omega_i}), \text{ where } m_\Omega(h) = m_{\omega_i}(h) \tag{6.12}$$

Let $T' \subseteq T$ be the subset of all positions in $T$ that are unambiguous with respect to the opponent types that selected $\Omega$'s moves at those positions. The likelihood for all positions in $T'$ together (assuming that the observations are independent) is:

$$\mathrm{L}(T'|\bar{\pi}) = \prod_{h \in T'} \mathrm{L}(m_\Omega(h) \,|\, \bar{\pi}) = \prod_{i=0}^{n-1} (\bar{\pi}_i \Pr(h_{\omega_i}))^{N_i} \tag{6.13}$$

where $N_i = \#\{h \in T' \,|\, m_{\omega_i}(h) = m_\Omega(h)\}$. The value of $\bar{\pi}$ for which the likelihood is maximal can easily be determined on the basis of the logarithm of the likelihood:

$$\begin{aligned} \ln \mathrm{L}(T'|\bar{\pi}) &= \sum_{i=0}^{n-1} N_i \ln \bar{\pi}_i + N_i \ln \Pr(h_{\omega_i}) \\ &= N_0 \ln(1 - \sum_{i=1}^{n-1} \bar{\pi}_i) + \sum_{i=1}^{n-1} N_i \ln \bar{\pi}_i + \sum_{i=0}^{n-1} N_i \ln \Pr(h_{\omega_i}) \end{aligned} \tag{6.14}$$

thereby taking into account that $\bar{\pi}_0 = 1 - \sum_{i=1}^{n-1} \bar{\pi}_i$. Since the function is smooth and continuous on the domain, the optimal value for $\bar{\pi}$ occurs when all partial deriviates of this function are zero:

$$\frac{\partial \ln \mathrm{L}(m_\Omega(T)|\bar{\pi})}{\partial \bar{\pi}_i} = 0 \Rightarrow \frac{N_i}{\bar{\pi}_i} - \frac{N_0}{1 - \sum_{i=1}^{n-1} \bar{\pi}_i} = 0 \Rightarrow \frac{N_i}{\bar{\pi}_i} = \frac{N_0}{\bar{\pi}_0} \tag{6.15}$$

This set of equations implies that the MLE $\hat{\bar{\pi}}$ for $\bar{\pi}$ is given by the proportions of the $N_i$'s:

$$\hat{\bar{\pi}}_i = \frac{N_i}{\sum_j N_j} = \frac{\#\{h \in T | m_{\omega_i}(h) = m_\Omega(h)\}}{\#T} \tag{6.16}$$

To conclude, off-line learning of opponent-type probabilities can be done by first determining for all positions in $T$, the opponent types that select the same move as $\Omega$, then removing all positions in which not exactly one opponent type agrees with $\Omega$ and, finally, computing the proportion for every opponent type. The resulting proportions are maximum-likelihood estimators of the opponent-type probabilities.

It is important to realize that the statistical method assumes that the real opponent $\Omega$ is using a mixed strategy of the same opponent types, an assumption that is inevitable. This method does not depend on correlations between opponent types or on the probability distribution over positions, although it is assumed that the set $T$ is representative for the set of positions encountered in real games. Furthermore, the size of the set $T'$ is important for the quality of the learned probabilities. The number of unambiguous moves in $T$ might be small if the branching factor of the game is low or if the opponent types are highly correlated on the positions in $T$. Also when one of the opponent-type probabilities is low, the number of positions in $T'$ for that opponent type might be too small to obtain a reliable estimation.

## 6.6.2 On-line learning of probabilities

The learning of opponent-type probabilities during a game is limited since the number of observations is low. It can, however, be useful to adapt probabilities that were achieved earlier, for instance by off-line learning. Two types of on-line learning can be distinguished: a *fast* one in which only the best move of every opponent type is used, and a *slow* one in which the search value of all moves is computed for all opponent types.

Fast on-line learning happens straightforwardly as follows: start with the prior obtained probabilities. At every move of the opponent do the following: for all opponent types detect whether their best move is equal to the actually selected move. If so, reward that opponent type with a small increase of the probability. If not, punish the opponent type. The size of the reward or punishment should not be too large because this type of learning will lead to the false supremacy of one of the opponent types. This type of incremental learning is also applied in the prediction of user actions (Davison and Hirsh, 1998).

Slow on-line learning would be an application of the naive Bayesian learner (see Domingos and Pazzani, 1997). A similar approach is used in learning probabilistic user profiles (Pazzani and Billsus, 1997). Slow on-line learning works as follows. For all opponent types $\omega_i$, the sub-game values $v_{\omega_i}(h + m_j)$ of all possible moves $m_j$ at position $h$ are computed. These values are transformed into conditional probabilities $\Pr(m_j|\omega_i)$, that indicate the "willingness" of the opponent type to select that move. This transformation can be done in a number of ways. An example is the method by Reibman and Ballard (see subsection 2.1.3): first determine the rank $r(m_j)$ of the moves according to $v_{\omega_i}(h+m_j)$ and then assign probabilities:

$$\Pr(m_j|\omega_i) = \frac{(1-P_s)^{r(m_j)-1} \cdot P_s}{\sum_k (1-P_s)^{r(m_k)-1} \cdot P_s} \tag{6.17}$$

$P_s$ ($\in (0,1]$) can be interpreted as the likeliness of the opponent type not to deviate from the best move: the higher $P_s$, the higher the probability on the best move. It is however also possible to use the actual values of $v_{\omega_i}(h+m_j)$. Now Bayes' rule is used to compute the opponent-type probabilities given the observed move of the opponent.

$$\Pr(\omega_i|m_\Omega(h)) = \frac{\Pr(m_\Omega(h)|\omega_i)\Pr(\omega_i)_t}{\sum_k \Pr(m_\Omega(h)|\omega_k)\Pr(\omega_k)_t} \tag{6.18}$$

These a-posteriori probabilities are used to update the opponent-type probabilities.

$$\Pr(\omega_i)_{t+1} = (1-\gamma)\Pr(\omega_i)_t + \gamma \Pr(\omega_i|m_\Omega(h)) \tag{6.19}$$

In this formula, parameter $\gamma$ ($\in [0,1]$) is the learning factor: the higher $\gamma$, the more influence the observations have on the opponent-type probabilities. The approach is called *naive* Bayesian learning, because the last formula assumes that the observations at the subsequent positions in the game are independent.

### 6.6.3 Learning opponent types

The second part of the probabilistic opponent model is the set of opponent types themselves. There are two basic approaches to the learning of individual opponent types: (1) to learn an evaluation function, a move ordering, the search depth and other search preferences used by a given opponent, and (2) to learn the opponent's strategy, which means to learn directly the move that the opponent selects at every position.

The first approach has been studied in computer chess, especially the learning of evaluation functions. Although the goal often is to obtain a good evaluation function for $\alpha$-$\beta$ search, similar techniques can be used for obtaining the evaluation function of an opponent type. Anantharaman (1997), for instance, describes a method to learn or tune an evaluation function with the aid of a large set of positions and the moves selected at those positions by master-level human players. The core of the approach is to adapt weights in an evaluation function by using a linear discriminant method in such a way that a certain score of the evaluation function is maximized. The evaluation function is assumed to have the following form: $V(h) = \sum_i W_i\, C_i(h)$. The components $C_i(h)$ are kept constant, only the weights $W_i$ are tuned. The method was used to tune an evaluation function for the program DEEP THOUGHT, a predecessor of DEEP BLUE. Although the method obtained a better function than the hand-tuned evaluation function of the program, the author admits that it is difficult to avoid local maxima. Fürnkranz (1996) gives an overview of machine learning in computer chess, including several other methods to obtain evaluation functions from move databases.

Direct learning of opponent strategies is studied extensively on repeated games (Fudenberg and Levine, 1998). These are (mostly) small games in which all players move simultaneously and immediately receive a payoff. The number of moves typically is small but the game is played many times. The goal is to maximize the payoff over time. Examples of repeated games are iterated prisoners dilemma and roshambo (rock-paper-scissors). Opponent modelling is important in these games, but the game complexity is so low that game-tree search is not an issue. This means that the opponent's strategy is modelled directly. Since the game is repeated, a player's strategy is not limited to one game but extends over multiple (even infinite) games. For learning opponent strategies in roshambo we refer to Egnor (2000). General learning in repeated games is studied, for example, by Carmel and Markovitch (1999).

A particular complex repeated game is poker. Billings *et al.* (2000) describe a poker-playing program, Poki, that incorporates an opponent-modelling component: "No poker strategy is complete without a good opponent modelling system. A strong poker player must develop a dynamically changing (adaptive) model of each opponent, to identify potential weaknesses." Opponent modelling is used with two distinct goals: to predict the next move of each opponent and to estimate the strength of each opponent's hand. The program uses a neural network for these predictions.

The applicability of these learning techniques for obtaining opponent types is limited since the nature of board games is too different from repeated games like roshambo and poker.

## 6.7 Chapter Conclusions

In this chapter we introduced a new approach for using an opponent model in game-tree search: Probabilistic Opponent-Model search. The approach is based on a probabilistic opponent model that constitutes a mixed strategy of a number of known opponent types. One of the opponent types is the player herself - a mechanism that should prevent the negative effects of overestimations.

PrOM search can be implemented in two forms: a one-pass version and a version with $\alpha$-$\beta$ probing. A number of standard search enhancements can be applied to PrOM search. The computational complexity of PrOM search is much higher than OM search in the versions with $\alpha$-$\beta$ probing but especially in the one-pass versions. This is mainly caused by the impossibility of deep pruning in PrOM search.

Opponent models for PrOM search can be learned from observations of the opponent. We concentrated on the learning of the opponent-type probabilities in an off-line and on-line setting.

Experiments with PrOM search will indicate whether this method indeed performs better than OM search and whether the learned opponent models are indeed nearly optimal.

# Chapter 7

# Experiments in PrOM Search

*To show the power of probabilistic Opponent-Model search we offer the reader a second proverbial pudding to be eaten. Like we did with OM search, we tested the performance of the search method in some actual games. We start with describing experiments on random game trees that aimed to gain insight into the average case for the computational efficiency. Experiments in the game lines of action showed that PrOM search can outperform OM search as well as Minimax. Off-line learning opponent-type probabilities was tested in a simulated game and in the game of bao. Finally, experiments in bao revealed that PrOM search, with imperfect knowledge of the opponent, sometimes performed better than Minimax. Although the difference was statistically significant, it was still small in bao.*

---

**Chapter contents:** This chapter describes four series of experiments with PrOM search. Section 7.1 names the four domains. Section 7.2 is dedicated to random game trees, section 7.3 to experiments in lines of action, section 7.4 to experiments with off-line learning in a simulated game, and section 7.5 to experiments in the game of bao. The chapter ends with conclusions in section 7.6.[1]

## 7.1 Four Different Domains

This chapter is, like chapter 5, an empirical follow-up of the previous theoretical chapter. Again we performed four series of experiments, in four different domains,

[1]Parts of this chapter have been published in Donkers, H.H.L.M., Uiterwijk, J.W.H.M., Herik, H.J. van den (2001), Probabilistic Opponent-Model Search, *Information Sciences*, Vol. 135, Nos. 3–4, pp. 123–149.

each concentrating on a different facet of PrOM search. Three of the domains (e.g., random game trees, the game LOA, and the game bao) are identical to domains in chapter 5, so there is no need to introduce those domains in this chapter. The fourth domain is the learning of opponent-type probabilities for which we refer to chapter 6.

In section 7.2 we concentrate on the analysis of the computational complexity of different implementations of PrOM search applied to random game trees in order to gain insight into their average-case complexities. Experiments in LOA are described in section 7.3. These experiments hinted that PrOM search can perform better than OM search and sometimes also better than Minimax.

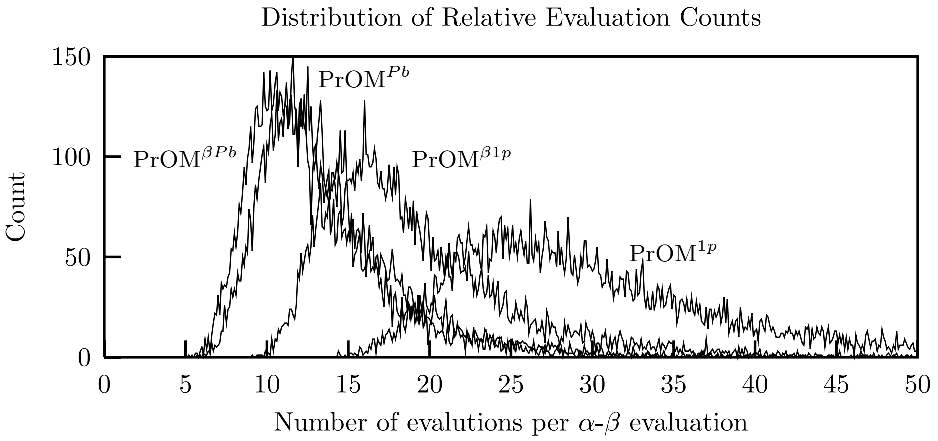
Section 7.4 describes experiments with off-line learning of opponent-type probabilities in a simulated game. The experiments in section 7.5 are used to study in detail the effectiveness of PrOM search in the game of bao. First, opponent-type probabilities are learned for different probabilistic opponent models. Then one probabilistic opponent model is studied in detail. Finally, the effectiveness of PrOM search in bao is investigated for a range of different opponent models. We end the chapter with a discussion and concluding remarks in section 7.6.

## 7.2 Experiments with Random Game Trees

The first series of experiments with PrOM search were performed on the same type of random game trees as introduced in section 5.2. The experiments were intended to provide insight into the average computational complexity of the different implementations of PrOM search presented in chapter 6. We conducted three experiments similar to the three experiments in section 5.2. In the first experiment (subsection 7.2.1) we concentrated on the distribution of the relative number of evaluations that the different implementations for PrOM search needed in comparison to $\alpha$-$\beta$ search. The second experiment (subsection 7.2.2) measured the influence of search depth, branching factor, and number of opponent types on the number of evaluations in PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$. The third experiment (subsection 7.2.3) concentrated on the improvement that transposition tables can cause in PrOM$^{\beta Pb}$. An additional experiment (subsection 7.2.4) deals with the PrOM-search value.

### 7.2.1 Distribution of evaluation counts

As an extension to the experiment in subsection 5.2.2, the 10,000 random game trees from that experiment were also used as input to four different implementations of PrOM search: PrOM$^{\beta 1p}$, PrOM$^{1p}$, PrOM$^{\beta Pb}$, and PrOM$^{Pb}$ (i.e., a version with and without $\beta$ pruning for one-pass PrOM search as well as for PrOM search with probing). Figure 7.1 shows the results. In the figure the distributions for OM$^{\beta 1p}$ and OM$^{\beta Pb}$ are repeated from figure 5.3. The figure shows that the effect of $\beta$ pruning was much larger for one-pass PrOM search than for PrOM search with $\beta$ probing. Furthermore, the relative number of evaluations in one-pass PrOM search, even with $\beta$ pruning, was larger than the number of evaluations needed in PrOM search with probing. The figure also shows that the spreading increased from OM search via probing PrOM search to one-pass PrOM search. In the sequel we will not consider



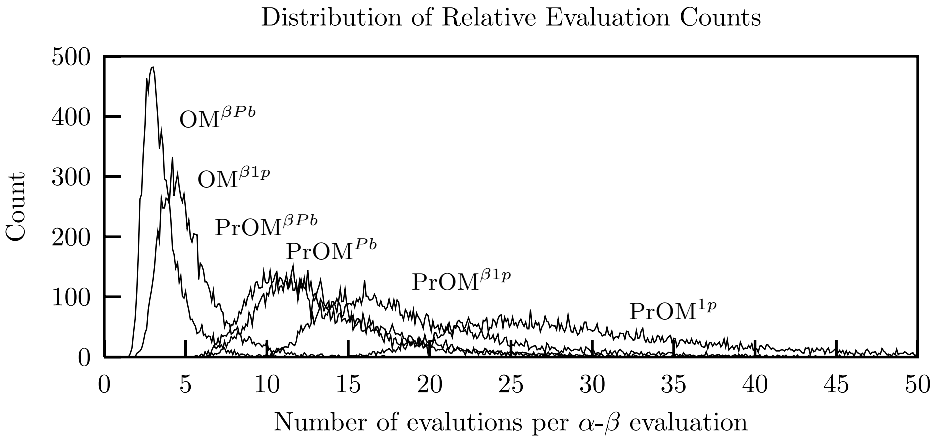


Figure 7.1: Distributions of the number of evaluations needed in 10,000 random game trees with depth 8 and width 4 for PrOM$^{\beta Pb}$, PrOM$^{Pb}$, PrOM$^{\beta 1p}$, and PrOM$^{1p}$, *divided* by the number of evaluations needed by $\alpha$-$\beta$ on the same trees. PrOM search used a opponent model with 2 opponent types. In the bottom figure, the graphs are repeated together with the data for OM search, repeated from figure 5.3.

non-pruning versions of PrOM search anymore and will concentrate on the differences between PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$.

### 7.2.2 Influence of the game-tree size

In order to study the relation between the size of the search tree and the computational complexity of PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$ in random game trees, we conducted three series of experiments. In the first series we varied the depth of the search trees, in the second series we varied the branching factor of the trees, and in the third series we varied the number of opponent types in the probabilistic opponent model. Per value of the search depth, branching factor, and number of opponent types, a sample of 100 random game trees was taken for which the number of evaluations that PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$ needed was determined. For every random game tree we also determined the number of evaluations that $\alpha$-$\beta$ search needed.

Figure 7.2 shows the results for the influence of the search depth. Since the number of evaluations appeared to behave exponentially in the search depth, we use a logarithmic scale for the y-axis. The resulting lines are almost linear, which confirms the exponential behaviour. The graphs show that for larger search depths PrOM$^{\beta 1p}$ needed more evaluations than PrOM$^{\beta Pb}$ on these random game trees. We noticed that the spread as indicated by the dotted lines lies almost symmetrical around the centre lines in this logarithmic scale.

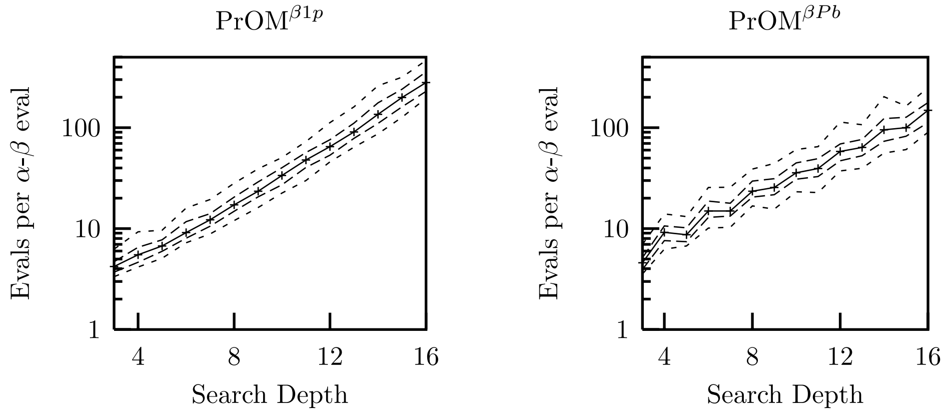


Figure 7.2: Relative number of evaluations needed by PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$ as a function of the search depth. The branching factor is 3, there are three opponent types, and the sample size is 100. The solid line in the middle is the median of each sample, the inner dashed lines represent the lower and upper quartiles, and the outer dotted lines represent the 5th and 95th percentile of the samples.

The influence of the branching factor is shown in figure 7.3. The figure clearly shows that PrOM$^{\beta 1p}$ needed more evaluations than PrOM$^{\beta Pb}$ on these random game trees. Both algorithms show a more-or-less linear relation between the number of evaluations per $\alpha$-$\beta$ evaluation and the branching factor.

The next figure, 7.4, depicts the influence of the number of opponent types. It shows no significant difference between PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$. Both algorithms behaved almost linearly in the number of opponent types.

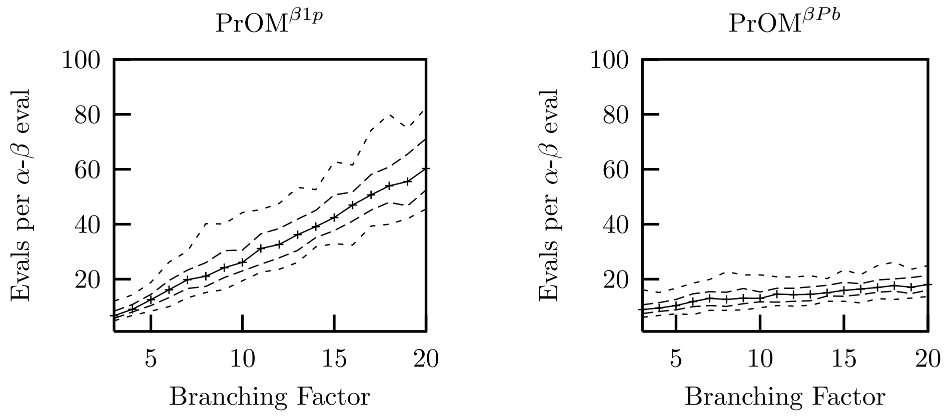


Figure 7.3: Relative number of evaluations needed by PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$ as a function of the branching factor. The search depth is 5, there are three opponent types, and the sample size is 100. The meaning of the lines is the same as in figure 7.2.

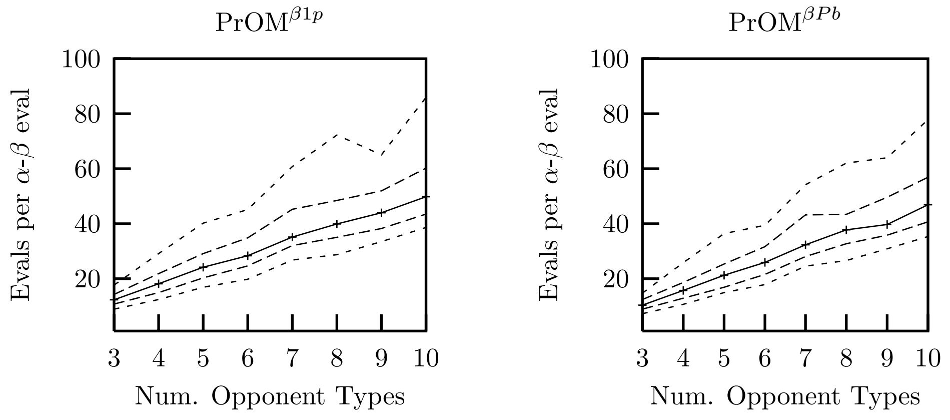


Figure 7.4: Relative number of evaluations needed by PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$ as a function of the number of opponent types. The search depth is 5, the branching factor is 3, and the sample size is 100. The meaning of the lines is the same as in figure 7.2.

### 7.2.3 Effect of transposition tables

In the next experiment on random game trees, we tested the effect of transposition tables in PrOM$^{\beta Pb}$. We provided every opponent type with a separate transposition table of $2^{17}$ entries. We used the same random game trees as in the previous subsection and measured the decrease in number of evaluations for each tree. No iterative deepening was used. Figure 7.5 shows the result of the experiment. It appears that the improvement by transposition tables increased with larger search depth, but at a given point (search depth 9 - 10), the transposition tables became overloaded and further improvement was not possible. With increasing branching factor, the improvement by transposition tables decreased. No overloading of the tables was visible here. When the number of opponent types was increased, the improvement by the tables increased slightly.

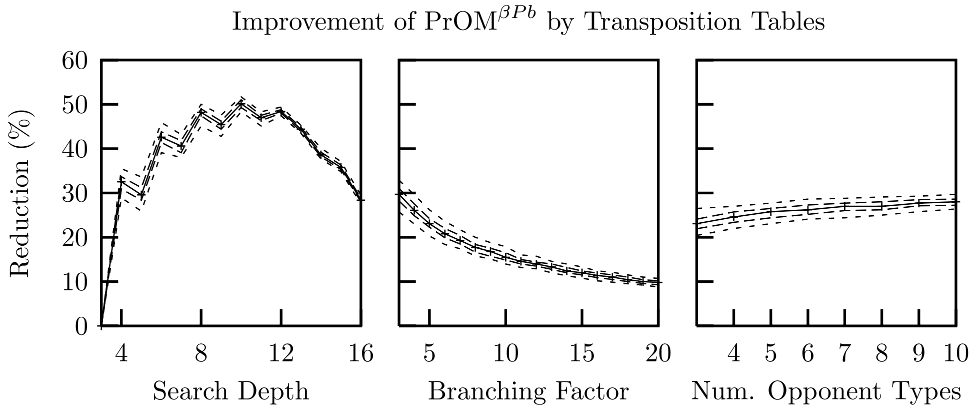


Figure 7.5: Improvement caused by transposition tables in PrOM$^{\beta Pb}$, as a function of the search depth, the branching factor, and the number of opponent types. The graph shows the percentage of evaluations that is ignored when the transposition table is used. The default search depth is 5, the default branching factor is 3, and the default number of opponent types is 3.

### 7.2.4 PrOM-search value

Figure 7.6 shows the result of a small experiment in which we wanted to investigate whether the PrOM-search value can be larger than the OM-search value with respect to one of the opponent types. In the experiment we used two opponent types, $\omega_0$ and $\omega_1$. We computed for every one of 10,000 trees the Minimax value with respect to $\omega_0$, the OM search value for $V_0 = \omega_0$ and $V_{op} = \omega_1$, and the PrOM-search value for the opponent model $[\omega_0, \omega_1]$ with probabilities (0.4, 0.6). In the figure we plotted the ratio between the value-surplus of PrOM search and the value-surplus of OM search. The figure shows that there are values above 1 which indicated that for some trees the PrOM search value indeed was higher than the OM-search value. The peak at 0

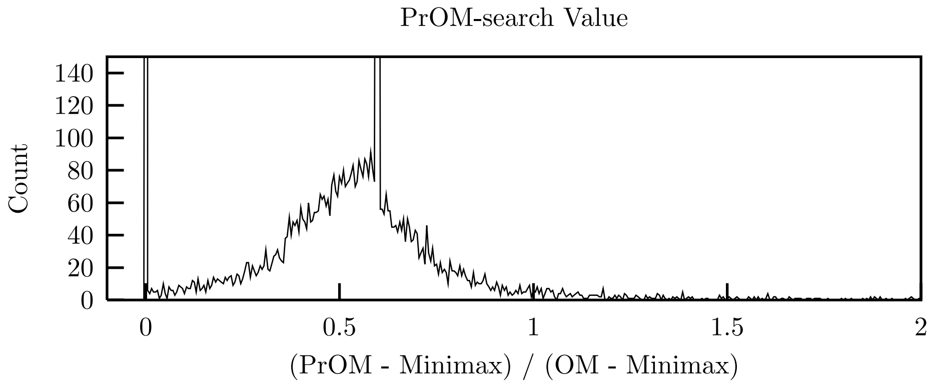


Figure 7.6: Histogram of the PrOM-search value surplus in relation to the OM-search value surplus of the same random game trees. There are two opponent types with probabilities (0.4, 0.6). The x-value 0 indicates the Minimax-value, the x-value 1 indicates the OM-search value. The game trees have depth 5 and branching factor 2. The sample size is 10,000.

represents the cases in which either the PrOM-search value or the OM-search value was equal to the Minimax value. The peak at 0.6 is an artefact that was produced by the branching factor of 2 in combination with the opponent-type probabilities[2].

## 7.2.5 Discussion

In line with the best-case analysis, the experiments with random game trees indicate that the probing versions of PrOM search performed better than the one-pass versions in this setting. Both PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$ behaved exponentially in the search depth, which is a serious obstruction for the practical applicability of PrOM search. Fortunately, PrOM$^{\beta Pb}$ can be improved considerably by transposition tables, but in case of larger search depths, the transposition table will be vulnerable to overloading.

Similar to the discussion in subsection 5.2.5, it should be noted that since the evaluation functions in random game trees are less correlated than in real games, the computational complexity results for the random game trees must be viewed more as a 'worst-case' scenario than as an 'average case'.

[2]The artefact is explained as follows. Let node $\boldsymbol{a}$ be at ply 1 on the principal variation. Assume that $\omega_0$ selects node $\boldsymbol{b}$ and $\omega_1$ selects node $\boldsymbol{c}$ at node $\boldsymbol{a}$. Assume further that $v(\boldsymbol{b}) = v_0(\boldsymbol{b}) = v_{mmx}$ and $v(\boldsymbol{c}) = v_{om}$. The PrOM-search value of the root in that case is $v_{prom} = 0.4v_{mmx} + 0.6v_{om}$. This leads to a value of 0.6 in the histogram: $(v_{prom} - v_{mmx})/(v_{om} - v_{mmx}) = (-0.6v_{mmx} + 0.6v_{om})/(v_{om} - v_{mmx}) = 0.6$.

## 7.3 Experiments in Lines of Action

The experiments with PrOM search in the game of LOA were performed alongside with the experiments in section 5.3. The set-up of the experiments therefore was almost the same as described in subsection 5.3.3. The difference was that in every tournament, every position was played again with three different settings for the first player. In all three settings, the first player used PrOM search with two opponent types: $\omega_0$ which is characterized by evaluation function $V_1$ and $\omega_1$ which is characterized by evaluation function $V_2$. The three settings varied in the opponent-type probabilities using opponent model $[\omega_0, \omega_1]$ with probabilities $(0.25, 0.75)$, $(0.5, 0.5)$, and $(0.75, 0.25)$, respectively. Tables 7.1, 7.2, and 7.3 show the results for the three tournaments A, B, and C. In the tables we repeated the results of the control matches from the tables in section 5.1.

Tournament A was a success for PrOM search. In every match except one, PrOM search played better than $\alpha$-$\beta$ search. Tournament B was no success: PrOM search played worse than $\alpha$-$\beta$ search (but not as bad as OM search). The last tournament (C) in which the evaluation functions were swapped, turned out to be a success for PrOM search again: in all but three matches, PrOM search played better (less worse) than $\alpha$-$\beta$ search.

In all three tournaments, the best results were obtained with the opponent-type probabilities $(0.5, 0.5)$. The fact that PrOM search played better than OM search in these experiments could be caused by the incorporation of the own evaluation function ($V1$) as opponent type $\omega_0$ in the opponent model. This could have compensated for the absence of admissibility in the pairs of evaluation functions. The tables show that roughly 50 percent of the probability had to be assigned to the own evaluation function for the best results.

Table 7.1: Results of the first LOA tournament. The numbers indicate the result over 60 games for the first player.

Tournament A: $V_1 = V_{NNa}$, $V_2 = V_{NNb}$

| Search | PrOM - $\alpha$-$\beta$ | | | |
|---|---|---|---|---|
| Depth | $\Pr(\omega_0) = 0.25$ | $\Pr(\omega_0) = 0.5$ | $\Pr(\omega_0) = 0.75$ | $\alpha$-$\beta$ - $\alpha$-$\beta$ |
| 2 | +20 | +46 | +42 | +18 |
| 3 | +20 | +8 | −12 | +4 |
| 4 | +22 | +38 | +14 | +6 |
| 5 | +4 | +12 | −4 | −20 |

Table 7.2: Results of the second LOA tournament. The numbers indicate the result over 60 games for the first player.

Tournament B: $V_1 = V_{NNa}$, $V_2 = V_c$

| Search | PrOM - $\alpha$-$\beta$ | | | |
|---|---|---|---|---|
| Depth | $\Pr(\omega_0) = 0.25$ | $\Pr(\omega_0) = 0.5$ | $\Pr(\omega_0) = 0.75$ | $\alpha$-$\beta$ - $\alpha$-$\beta$ |
| 2 | −4 | +2 | +4 | −4 |
| 3 | +8 | +22 | 0 | +24 |
| 4 | +6 | +8 | +2 | +18 |
| 5 | +4 | 0 | −2 | +24 |

Table 7.3: Results of the third LOA tournament. The numbers indicate the result over 60 games for the first player.

Tournament C: $V_1 = V_c$, $V_2 = V_{NNa}$

| Search | PrOM - $\alpha$-$\beta$ | | | |
|---|---|---|---|---|
| Depth | $\Pr(\omega_0) = 0.25$ | $\Pr(\omega_0) = 0.5$ | $\Pr(\omega_0) = 0.75$ | $\alpha$-$\beta$ - $\alpha$-$\beta$ |
| 2 | +12 | −4 | −4 | +4 |
| 3 | −24 | −20 | −12 | −24 |
| 4 | −10 | +10 | −8 | −18 |
| 5 | −20 | −4 | −28 | −24 |

## 7.4 Experiments in Off-line Learning

In this section we describe experiments designed to investigate off-line learning of opponent-type probabilities as described in subsection 6.6.1. The experiments in this section are performed in a simulated game environment. In the simulations, moves are selected by means of a predefined probability distribution, not by game-tree search. In subsection 7.5.1 we will also describe experiments with off-line learning in the real game bao.

### 7.4.1 Known opponent types

Since the goal of off-line learning is to find a probability distribution that predicts the moves of a target opponent as good as possible, game playing was simulated in the experiments by simply picking a number out of a given range of numbers. This number represented a move that supposedly was played.

We performed three experiments. We will discuss the first two experiments simultaneously in this subsection and will discuss the third experiment in subsection 7.4.2. In the first experiment, the target opponent $\Omega$ used a mixed strategy which consisted of opponent types $\omega_0, \ldots, \omega_n$ and probabilities $\Pr(\omega_i)$. The strategy of each opponent type $\omega_i$ was to pick a number $m_i$ randomly from the set $\{0, \ldots, M-1\}$ with equal probabilities for every number. This means that the choices of the opponent types were independent. In the second experiment, the strategies of the opponent types were not independent. The first opponent type ($\omega_0$) selected a number $m_0$ using with equal probability for all $M$ moves. Opponent type $\omega_i$, $i > 0$, selected a move from the set $\{(m_{i-1} + k) \mod M \mid k \in [-4, 4]\}$. In both experiments, the opponent types and there choices were known to the learning player.

The simulations were conducted as follows. Every time step $(1 \ldots \mathrm{T})$ consisted of three actions: (1) all opponent types $\omega_i$ selected a number $m_i$, (2) the target opponent $\Omega$ selected one opponent type $\omega_\Omega$, according to $\Pr(\omega_\Omega)$ and (3) $\Omega$ produced the number, $m_\Omega$, that was selected by $\omega_\Omega$. The observed $m_i$'s and $m_\Omega$'s led to an estimation $\hat{\Pr}(\omega_i)$ of $\Pr(\omega_i)$ by counting the number of times $N_i$ that only move $m_i$ agreed with $m_\Omega$ and divided this by the sum of the numbers of times $N_j$ that only move $m_j$ agreed with $m_\Omega$ (according to formula 6.16 in subsection 6.6.1). As a measure of the quality, a learning error $\epsilon$ was introduced, being the Euclidian distance $\sqrt{\sum_i (\hat{\Pr}(\omega_i) - \Pr(\omega_i))^2}$. The range of $\epsilon$ was $[0 \ldots \sqrt{2}]$.

In both experiments, the number of opponent types $N$ varied from 2 to 20. The probability distribution $\Pr(\omega_0, \omega_1, \ldots, \omega_{N-1})$ of the opponent types was of the form $(a, b, \ldots, b)$ (where $a$ varied between 0 and 1, and $b = (1-a)/(N-1)$). The range of available numbers, $M$, was fixed to 20. The learning time T varied from $10^1$ to $10^5$ runs. The sample size was 100 everywhere.

Figures 7.7 shows the results for 5 opponents. The y-scale in these figures is logarithmic to indicate that the error is approaching zero with increasing learning time. It appears that the probabilities can be learned exactly, although in the case of dependent opponent types, there appears to be a slow-down in the decrease of the error. This difference can be explained by the number of observations that had to be disregarded because more than one opponent type agreed with $\Omega$'s choice.

In the experiments, the fraction of disregarded events with independent opponent types was constantly 18.6%; in the case of dependent opponent types, the fraction of disregarded events varied between 48.7% and 53.8%. The figures also show that the learning error decreases when $\Pr(\omega_0)$ approaches 1.

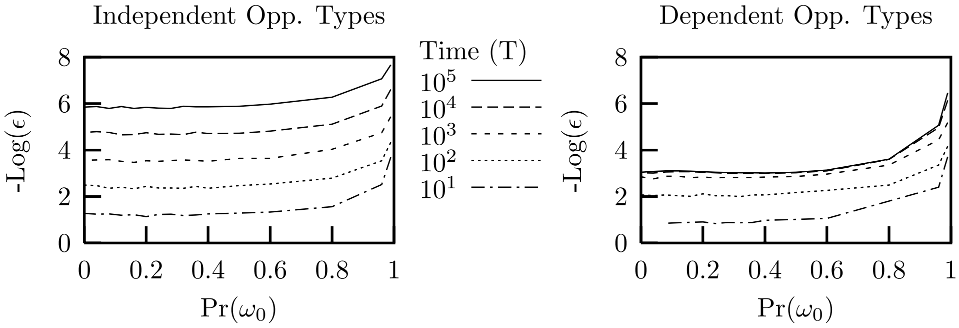


Figure 7.7: Average learning error $\epsilon$ for a set of 5 opponent types as a function of the opponent-type probabilities.

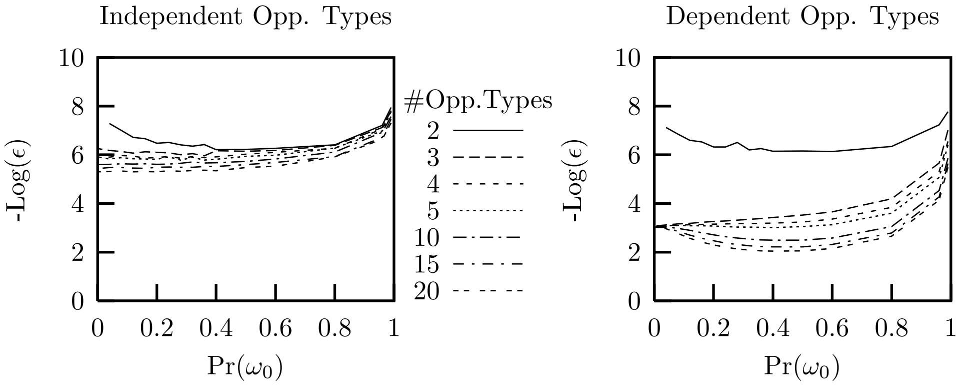


Figure 7.8: Average learning error $\epsilon$ as a function of opponent-type probabilities and the number of opponent types for T = $10^5$.

Figure 7.8 shows the results for varying numbers of opponent types. There appears to be a gap between the learning efficiencies of two and of more than two opponent types, especially at low probabilities. This gap is larger in the case of dependent types. The effect can be explained as follows. If there are only two opponent types, the learning error should be symmetrical in $\Pr(\omega_0)$: the error at $\Pr(\omega_0) = p$ should approximate the error at $\Pr(\omega_0) = 1 - p$. The figures show this symmetry for the case of two opponent types. However, such a symmetry is not required for the case of more than two opponent types. There is a peculiar, unexplained effect in the right graph of figure 7.8. The learning error for the cases of more than two opponent types seem to converge to $\epsilon = 10^{-3}$ when $\Pr(\omega_0)$ approaches 0.

### 7.4.2 Unknown opponent types

In the third experiment the learning player still assumed that $\Omega$ was using a mixed strategy as in the first two experiments, but $\Omega$ was in fact using another strategy. It simply picked the same move every time. The opponent types that we used in learning were the same ones as in the second experiment in subsection 7.4.1. The probability distribution that was being learned represented the player's approximative opponent model of $\Omega$. The goal of this experiment was to investigate whether a stable probability distribution was learned and how this distribution depends on the particular move that $\Omega$ selected all the time.

Since the correct probability distribution was unknown (unlike in the first two experiments), the Eucledian distance was not an appropriate measure of error, so we defined the learning error $\epsilon$ as the standard deviation over 100 samples. The experiment was performed with 5 opponent types only.

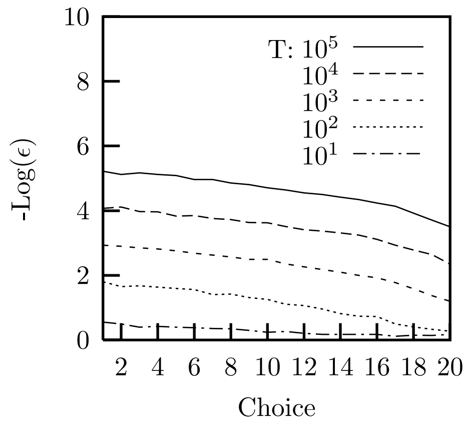


Figure 7.9: Average learning error $\epsilon$ as a function of opponent's choice for varying learning times.

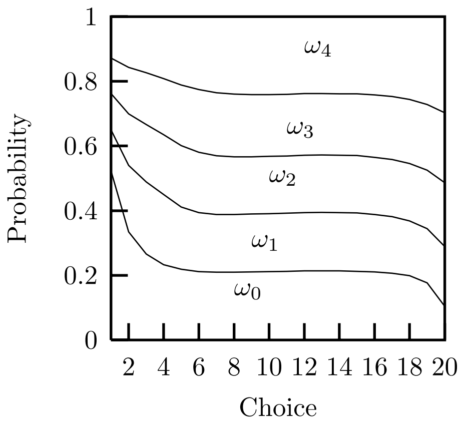


Figure 7.10: Average learned probability distribution over opponent types as a function of opponent's choice.

The way in which the errors decreased, as illustrated in figure 7.9, indicate that the learning resulted in stable probability distributions over the opponent types. Figure 7.10 shows the learned probability distributions. These are in concordance with the way in which the opponent types were defined. Since there are five opponent types and the shift between the move-selection windows for the opponent types was 4, Opponent type $\omega_0$ most probably selected low numbers and $\omega_4$ most probably high numbers. So, when $\Omega$ selects low numbers, $\omega_0$ agrees the most with $\Omega$ and when $\Omega$ selects high numbers, $\omega_4$ agrees the most.

### 7.4.3 Discussion

The experiments with off-line learning of opponent-type probabilities in a simulated game did not only confirm some of the expectations from the theoretical analysis, but also gave insights into the dynamics of the learning. It appeared that dependencies between opponent types can slow down the learning. The last experiment showed that an approximative probabilistic opponent model can be learned.

The experiments did not give any hint on the effect that the learned opponent models will have on the actual performance of PrOM search. In subsection 6.6.1 we hypothesized that the best opponent model for PrOM search is the model that predicts the moves of the opponent the best. The experiments in this section did not give an answer to this hypothesis. In the next section, we will therefore learn opponent models in a real game and test these models in PrOM search.

## 7.5 Experiments in Bao

We performed three experiments with PrOM search in the game of bao. The properties of this game are explained in subsection 5.5.2. In the first experiment (subsection 7.5.1) we used off-line learning to obtain the opponent-type probabilities for a range of opponent models. In the second experiment (subsection 7.5.2) we studied the performance of one opponent model in particular for different opponent-type probabilities. The last experiment (subsection 7.5.3) was a large tournament in which we studied the performance of a range of opponent models for different opponent-type probabilities.

In all experiments, the opponent used evaluation function NGND6A (see subsection 5.5.3). Except for some part of the third experiment, the probabilistic opponent models were based on MATERIAL, DEFAULT, GA3, and TDL2B. This means that the player using PrOM search did not have perfect knowledge of the opponent. This is in contrast with the experiments with OM search in bao as described in section 5.5.

### 7.5.1 Learning opponent-type probabilities

The first experiment with PrOM search in bao concentrated on learning opponent-type probabilities. The real opponent $\Omega$ used NGND6A and we learned opponent-type probabilities for all eleven combinations of two or more from MATERIAL, DEFAULT, GA3, and TDL2B. These evaluation functions all are (operationally) weaker than NGND6A, according to table 5.5.

The procedure that we applied for learning was as follows. At every time step we randomly generated a legal bao position. Then we used $\alpha$-$\beta$ search with search depth 4 to determine the move that $\Omega$ (NGND6A) would select and used $\alpha$-$\beta$ search with again depth 4 for every opponent type involved in the model to determine the moves selected by those types. When either none or more than one opponent type selected the same move as $\Omega$, the position was disposed. The process continues until 100,000 non-disposed positions were encountered. This sample size was chosen in order to ensure an estimation error of at most 0.3% at a confidence level of 95%.

The random bao positions were obtained by first picking a uniformly random number $k$ from the range $3 \ldots 30$, and then generating and executing $2k$ random but legal moves, beginning at the official start position of bao (see appendix B). If an end position was reached before $2k$ moves were executed, the position was discarded and the random generation of moves started all over.

Table 7.4: The probabilistic opponent models of the opponent NGND6A for all eleven combinations of the other opponent types. The last two columns give the number of positions generated and the percentage of positions used among these. The estimation error is everywhere smaller than 0.3% with a confidence level of 95%.

| | Opponent type | | | | | |
|---|---|---|---|---|---|---|
| Model | MATERIAL | DEFAULT | GA3 | TDL2B | Positions | Used |
| 1 | 49.28% | 50.72% | - | - | 351,124 | 28.48% |
| 2 | 42.49% | - | 57.51% | - | 483,092 | 20.70% |
| 3 | 33.04% | - | - | 66.96% | 410,509 | 24.36% |
| 4 | - | 45.80% | 54.20% | - | 343,171 | 29.14% |
| 5 | - | 36.93% | - | 63.07% | 338,066 | 29.58% |
| 6 | - | - | 40.06% | 59.94% | 415,282 | 24.08% |
| 7 | 20.86% | 47.10% | 32.04% | - | 579,710 | 17.25% |
| 8 | 18.76% | 35.72% | - | 45.52% | 583,431 | 17.14% |
| 9 | 17.47% | - | 28.28% | 54.25% | 644,745 | 15.51% |
| 10 | - | 33.47% | 25.68% | 40.85% | 608,643 | 16.43% |
| 11 | 10.53% | 34.61% | 18.17% | 36.69% | 845,309 | 11.83% |

Table 7.4 shows the learned opponent-type probabilities for the eleven opponent models. The first observation from this table is that the amount of disregarded positions increases with the number of opponent types involved. A second observation is that the probability of an opponent type is higher when that type received a higher score in table 5.5. An exception is the pair DEFAULT and GA3 in models 7, 10, and 11. But also in table 5.5, the relation between these two types is not straightforward.

### 7.5.2 Performance of TDL2B-DEFAULT versus NGND6A

In this subsection we study in detail the behaviour of one opponent model, consisting of the types TDL2B and DEFAULT, against the target opponent NGND6A in order to discover which circumstances influence the performance of PrOM search. We performed a series of 120,000 games between a player using PrOM search with the opponent model $[\omega_0 = \text{TDL2B}, \omega_1 = \text{DEFAULT}]$ and a player using $\alpha$-$\beta$ search with NGND6A. The games where divided in 12 groups of 10,000 games in which the opponent-type probability Pr(TDL2B) was set to 0, 0.1, 0.2, ..., 1, and 0.6307 respectively, the latter value being the probability that was learned in the previous subsection.

The setting of every game was as follows: PrOM search was used by South, $\alpha$-$\beta$ search by North. The search depth was 6 for both players. The start positions were generated exactly in the same way as the random positions in the previous subsection with $k$ ranging from 3 to 32. From every game we recorded (a) the winner, (b) the length of the game in plies, (c) the move number at which the game started (the number $k$), and (d) the search value of the first move by South. Below we present different views on the recorded data.

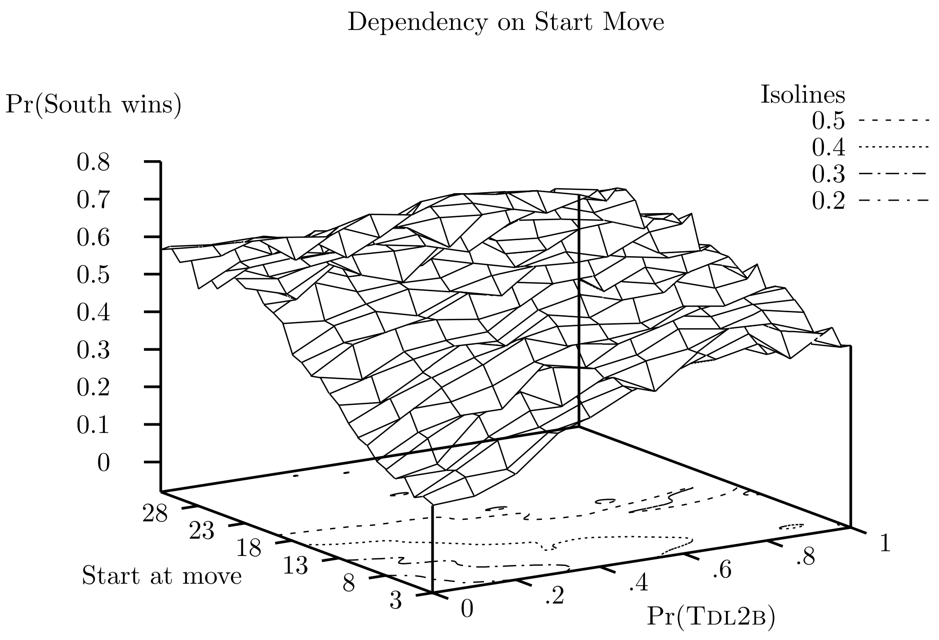


Figure 7.11: Win probability of PrOM search for the model TDL2B-DEFAULT versus NGND6A plotted against the start move of the game and the opponent-type probability of TDL2B. At the floor of the graph the isolines with equal win probability are presented.

Figure 7.11 shows the probability of South to win as a function of the move at which the games started and the opponent-type probability of TDL2B. The picture shows that for games that started early (moves 3-8) the win probability is generally lower than for games that started later (moves 18-32). It also shows that when games started earlier, the win probability depended more on Pr(TDL2B) than when games started later. At early start moves, the win probability clearly drops if Pr(TDL2B) approaches zero. In passing we note that for Pr(TDL2B) = 0, PrOM search is equal to OM search and for Pr(TDL2B) = 1, PrOM search is equal to Minimax search. The figure suggests that for Pr(TDL2B) = 0.8, the win probability is slightly higher than for Pr(TDL2B) = 1, which would mean that PrOM search is performing better than Minimax at that point.

To emphasize the dependency of the win probability on the start move, we present in figure 7.12 the win probability for South as a function of Pr(TDL2B) for three groups of start moves: (3-8), (9-19), and (20-32). The figure confirms that the win probability increases if the start move is later. Furthermore, the win probability is almost constant for start moves later than 20, and it is increasing almost monotone for the middle range of start moves. For the early start moves, there clearly is an optimum at Pr(TDL2B) = 0.8.

Before we attempt to explain the dependency of the win probability on the start move, we first present a second view on the data. In figure 7.13 the win probability for South as a function of Pr(TDL2B) is plotted for two sets of games: those that

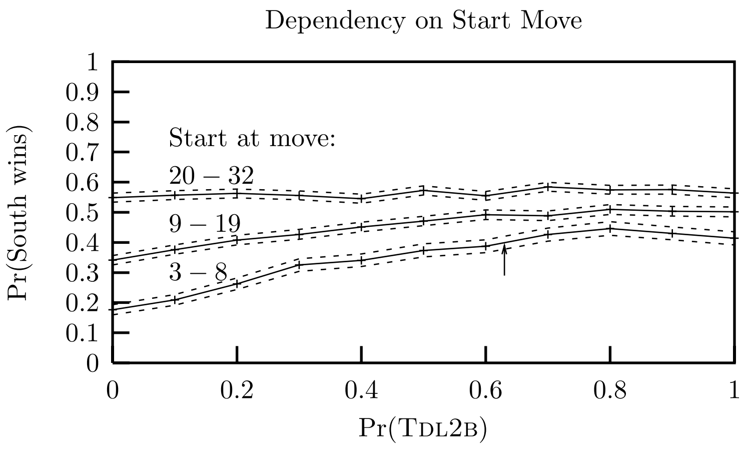


Figure 7.12: Win probability of PrOM search for the model TDL2B-DEFAULT versus NGND6A plotted against the opponent-type probability of TDL2B for three groups of start moves. The dotted lines give the 95%-confidence intervals. The arrow indicates the value of Pr(TDL2B) (0.6307) that was learned in subsection 7.5.1 and the win probability at that point.

end within 27 ply and those that end after 27 ply. The number 27 is chosen because it appeared to provide the most informative division. The effect is comparable to the results in figure 7.12. For long games, there is a clear dependency of the win probability on Pr(TDL2B) and an optimum appears at Pr(TDL2B) = 0.8. For short games, the dependency and the optimum are less clear.

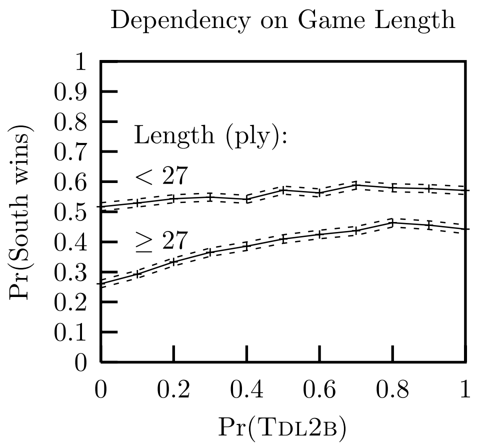


Figure 7.13: Win probability of PrOM search for the model TDL2B-DEFAULT versus NGND6A for long and short games.

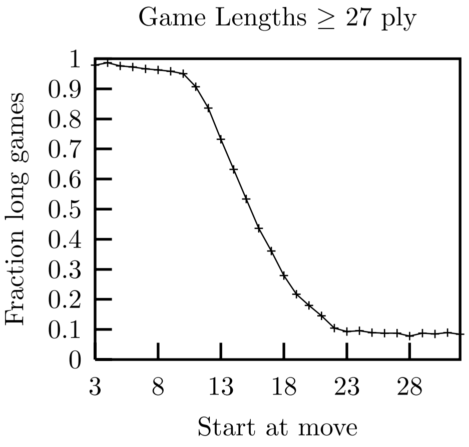


Figure 7.14: Fraction of long games (≥ 27 ply) as a function of the start move.

The resemblance between figures 7.12 and 7.13 suggests that the start move and game length are correlated. Figure 7.14 confirms this. At early start moves, almost all games last more than 27 ply, at late start moves, almost no game lasts more than 27 ply. This correlation can be explained by a special property of bao. The game of bao has two stages with different sets of rules. The second stage starts after exactly 22 moves (44 ply). Studies in random play revealed that most bao games last the whole first stage but are finished around 5 moves after the start of the second stage, which is 54 ply from the beginning. The sharp drop of long games begins at about start move 12 (ply 24). When 27 ply is added to this start, we obtain 51 ply. The drop in figure 7.14 ends at move 23 (ply 46) which is close to the 54 ply. This explains the relation between the start move and the game length.

The relation between the start move and the general height of the win probability is less obvious. It might be caused by the random generation of the positions that gave the South player an advantage over North at these positions, since the effect also occurs for Pr(TDL2B) = 1 (Minimax). The effect of the start move on the dependency of the win probability on Pr(TDL2B) can be explained by the fact that when the games started early, the length of the games was longer, which gave the PrOM search player more opportunities to speculate. Apparently, the effects of the speculations depended on the opponent-type probabilities.

The win probability at the optimum (Pr(TDL2B) = 0.8) for start moves earlier than 9 is statistically significant higher than the win probability at Pr(TDL2B) = 1 with a confidence of 95%. This means, stated carefully, that we cannot reject the hypothesis that PrOM search performs better at this point than Minimax on the basis of this experiment. However, the difference is not very large: 0.446 at Pr(TDL2B) = 0.8 against 0.413 at Pr(TDL2B) = 1. The win probability at Pr(TDL2B) = 0.6307 (the probability that was learned in the subsection 7.5.1) was 0.407, which is lower than the value for Minimax. This means that the optimal effect of PrOM search does not occur at the learned probability but at a higher value of Pr(TDL2B).

Another aspect of PrOM search and OM search is revealed by figures 7.15 and 7.16. In these figures we plotted the win probability against the score (search value) of the first move. The figures show the accuracy of the prognosis of this value. Since TDL2B is a weaker evaluation function than NGND6A, the graph for Pr(TDL2B) = 1 (Minimax) is not symmetric around zero but is shifted to the right. The win probability at score zero is about 0.4, which is not far from the result of table 5.5. The graph for Pr(TDL2B) = 0 (OM search) lies below that of Minimax. This illustrates the self-overestimation of OM search: at a score of zero, only 10% of the games is won. At score 80 above which Minimax wins all games, OM search only wins 50% of the games. This self-overestimation can be caused by type-I errors as described in subsection 4.2.1. Figure 7.16 shows that for the opponent-type probabilities between 0 and 1 there is a gradual transition from the Minimax graph to the OM-search graph. This indicates that the mixed-strategy model of PrOM search probably diminishes the effects of type-I errors; the higher Pr(TDL2B) the less self-overestimation occurs.

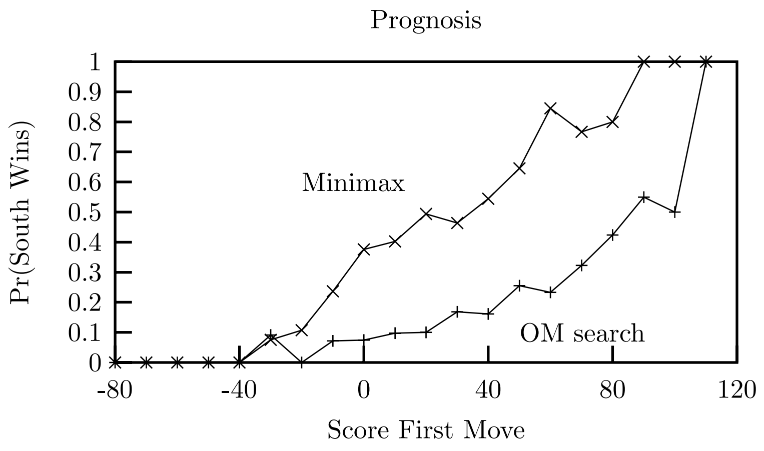


Figure 7.15: Win prognosis of Minimax and OM search compared. These are the two border-cases in figure 7.16 for the probabilities Pr(TDL2B) = 0 (OM search) and Pr(TDL2B) = 1 (Minimax search) respectively.

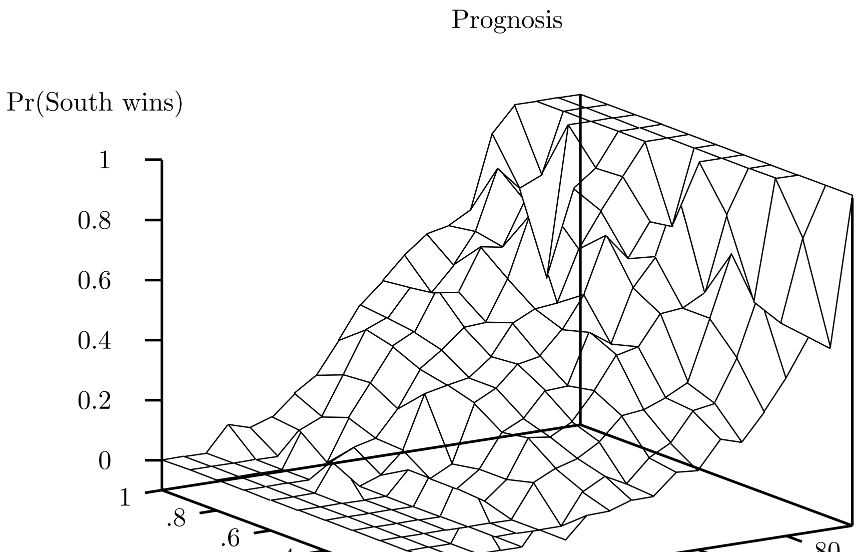


Figure 7.16: Win prognosis of PrOM search for the model TDL2B-DEFAULT versus NGND6A plotted against the the opponent-type probability of TDL2B and the PrOM-search value of the first move.

### 7.5.3 Grand tournament

The last experiment that we performed on the game of bao was a tournament between all opponent models constructed from combinations of two opponent types out of five against NGND6A. For every opponent model we played 2,000 games for each value 0, 0.1, 0.2, ..., 1.0 of $\Pr(\omega_0)$. PrOM search was used by South, $\alpha$-$\beta$ search with NGND6A was used by North. The search depth was 6 for both players. The start positions were randomly generated in the same way as in the previous subsections, but the range was restricted to start move 3 to 8. Figure 7.17 shows the results of this experiment. Every square contains the results for a different opponent model. The names of the two players in each opponent model are indicated in the squares. The first player constitutes $\omega_0$, the second player constitutes $\omega_1$. The squares are arranged in such way that on every row, $\omega_0$ is constant and on every column $\omega_1$ is constant, except for the diagonal. The squares on the diagonal show the results for the opponent models that have the true opponent, NGND6A, as $\omega_1$. On the vertical axis of each square is the win probability. On the horizontal axis of each square is the probability $\Pr(\omega_0)$. At probability $\Pr(\omega_0) = 0$ (at the left of each square) PrOM search is equivalent to OM search; at probability $\Pr(\omega_0) = 1$ (at the right of each square) PrOM search is equivalent to Minimax search. The arrows indicate the opponent-type probabilities learned in subsection 7.5.1. There are no arrows placed in the squares on the diagonal since $\omega_1$ is the true opponent in those opponent models and hence the opponent-type probabilities would be learned to be 0 for $\omega_0$. The dotted lines above and below the straight lines indicate the 95% confidence interval for the win probabilities.

The win probability at $\Pr(\omega_0) = 1$ (the point where PrOM search is equivalent to Minimax search) is the same in every square of each row since $\omega_0$ is the same at each row. The win probability at $\Pr(\omega_0) = 1$ is the highest at the bottom row (where $\omega_0$ is NGND6A).

In all squares the win probability is the lowest at $\Pr(\omega_0) = 0$, which is the probability where PrOM search is equivalent to OM search. It means that for all cases that $\Pr(\omega_0)$ was larger than zero, PrOM search performed better than OM search. This result agrees with our hypothesis that including the own evaluation function into the opponent model is beneficial.

There are four squares in which there is a probability $\Pr(\omega_0)$ smaller than 1 at which the win probability was higher than the win probability at $\Pr(\omega_0) = 1$ (where PrOM search is equivalent to Minimax search) with statistical significance. These are: DEFAULT-MATERIAL at $\Pr(\omega_0) = 0.9$, GA3-DEFAULT at $\Pr(\omega_0) = 0.8$, GA3-NGND6A at $\Pr(\omega_0) = 0.7$, and TDL2B-DEFAULT at $\Pr(\omega_0) = 0.8$. It means that in these cases, PrOM search performed better than Minimax search. Although an optimum above the value for Minimax is only statistically significant in those four cases, in all other cases, except one (GA3-MATERIAL), an optimum above the Minimax value is at least suggested by the graphs. However, the effect of PrOM search at these optima is small.

We make two further observations on the results. The first one is that optima all lie at probabilities considerably higher than the probabilities learned in subsection 7.5.1 (indicated by the arrows). This suggests that the learned opponent-type probabilities are not the probabilities with which PrOM search performs best: it seems that more probability should be assigned to the own evaluation function ($\omega_0$).

The second observation is that some of the graphs in figure 7.17 show a peculiarity that seems too regular to be explained by noise: there is a wave-like pattern in the graph of TDL2B-NGND6. The waves have a length of 0.3. Waves of the same length are visible in the graph of GA3-NGND6 and in the graph of DEFAULT-NGND6. Further research is needed to explain these waves.

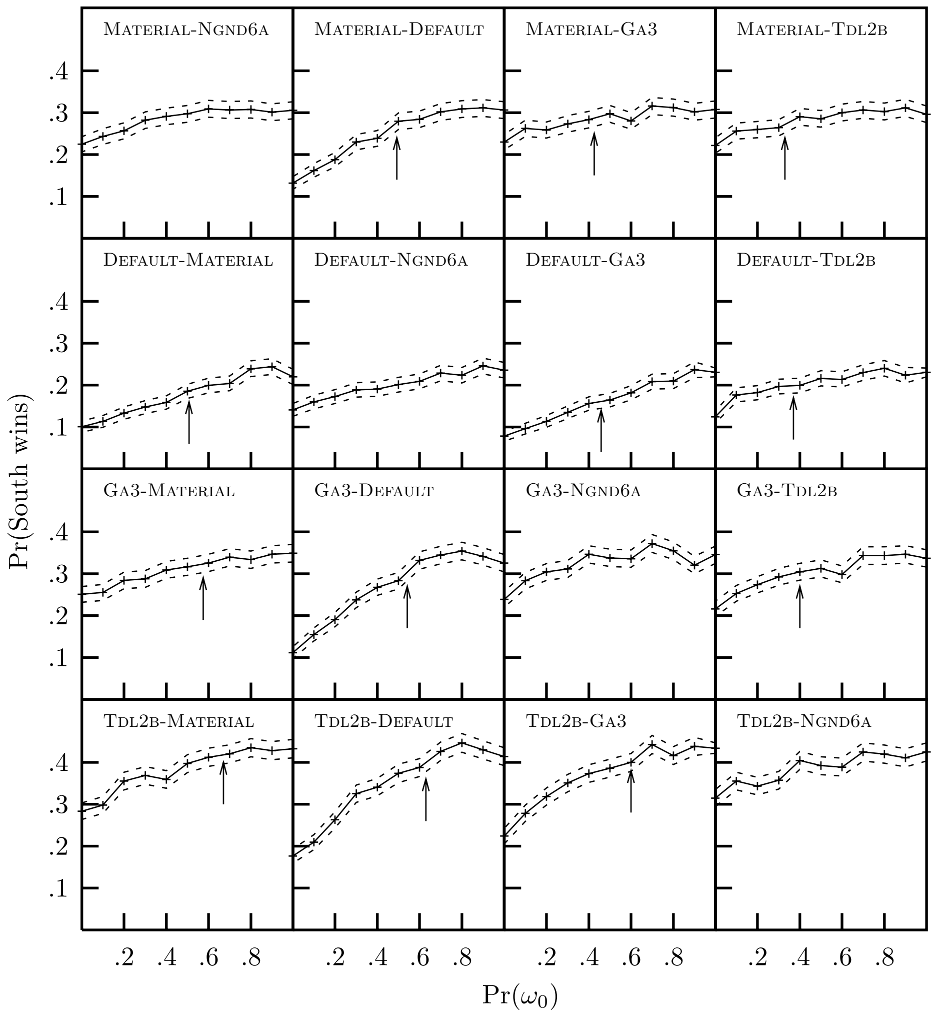


Figure 7.17: Overview of the performance of probabilistic opponent models in bao, expressed in South' win probability as a function of Pr($\omega_0$). The opponent models $\omega_0$-$\omega_1$ are indicated inside the boxes. The arrows indicate the probabilities learned in subsection 7.5.1. The dotted lines indicate the 95% confidence interval for the win probabilities.

### 7.5.4 Discussion

The experiments with PrOM search showed that PrOM search in all cases performed better then OM search. The experiments also showed that PrOM search can perform better in bao than Minimax even when no perfect knowledge of the opponent is available. However, the gain from using PrOM search is not very large. Furthermore, the search depth was kept constant to six for both sides, which means a considerable advantage for PrOM search since this search method evaluates much more leaf nodes than $\alpha$-$\beta$ search and therefore uses much more time for the same search depth.

The opponent-type probabilities that were learned off-line, appeared not to be the best probabilities to use in PrOM search: the probability $\Pr(\omega_0)$ had to be larger than the learned value. This suggests that the opponent model should put more weight on the player's own evaluation function than is needed for the prediction of the opponent's moves. A possible explanation for this is that the extra weight balances the negative effects of type-I errors.

## 7.6 Chapter Conclusions

The main conclusion that we can draw from the experiments in this chapter is that PrOM search performs better than OM search in the game of LOA as well as in bao. It also can perform better than Minimax. This is certainly so for LOA, but PrOM search also performs slightly better than Minimax in bao. The difference in performance between LOA and bao could be explained by the low branching factor of bao (3 to 5 on average). This property allowed us to play many games, but it possibly offered fewer opportunities to PrOM search for speculation. However, the large difference between the performance of Minimax and OM search in bao suggests that speculation occurs at a certain pace, sufficient to allow OM search to make many errors. Therefore more research is needed in these and other games to study the exact factors that influence the performance of PrOM search.

The fact that the opponent-type probabilities that are learned using the off-line procedure are not the ones that perform optimally, has as a consequence that on-line learning of opponent-type probabilities in PrOM search is not feasible in the way that is described in subsection 6.6.2. The procedure described will drive the probabilities to suboptimal values. Therefore, no experiments were performed with this type of on-line learning. A possible way to learn opponent-type probabilities in a correct way might be by using a form of reinforcement learning. This should be a subject of future research.

Finally we have to remark that the computational costs of PrOM search are very high, especially when the search depth increases. The application of transposition tables in PrOM$^{\beta Pb}$ can take some of the costs away, but this will not remove the exponential character of the ratio between PrOM-search costs and $\alpha$-$\beta$-search costs.

# Chapter 8

# Conclusions

*At the end of this thesis we return to the problem statement and research questions. Some of our research questions are answered in a positive way, but the main outcome is that there exists a great risk when using opponent models in search. This risk partly comes from sources that were already known, but in this thesis we discovered new causes of risk. One important cause of risk is the occurrence of estimation errors in the own evaluation function. The title of this thesis, Nosce Hostem, means 'know the opponent' and hints at the quality of the opponent models. Our analysis, however, suggests that Thales of Miletos' original maxim should be applied too: Nosce Te Ipsum (Γνῶθι Σαυτόν: know thyself).*

---

**Chapter contents:** In section 8.1, our four research questions are revisited. The problem statement is answered in section 8.2. We list our contributions and the significance of the thesis in section 8.3. The chapter ends with future research and recommendations in section 8.4.

## 8.1 Answers to the Research Questions

In section 1.5 we presented our problem statement and posed four research questions that should be answered before we could deal with the problem statement. In this section we answer these research questions one by one. In the next section we will formulate from these answers a reply to the problem statement.

### 8.1.1 Theoretical properties

The first research question posed was on the theoretical properties of OM search and PrOM search.

> **Research question 1**: *What are the important theoretical properties of Opponent-Model search and probabilistic Opponent-Model search?*

This research question was the subject of chapters 2, 4, and 6. In chapter 2, we collected the theoretical properties of OM search described previously. The first property was the relation between the OM search value and the Minimax value at the root of a search tree (Carmel and Markovitch, 1993; Iida *et al.*, 1993a). We showed in subsection 6.2.4 that this relation is valid for a whole range of search methods, including PrOM search. The importance of this property is limited since an OM search value that is higher than the Minimax value of a tree does not guarantee a higher win probability. Figure 7.15 illustrates that the OM-search value does not constitute a high-quality prognosis.

The second property of OM search previously described (Carmel and Markovitch, 1993; Iida *et al.*, 1993a) is the risk of OM search which is caused by imperfect knowledge of the opponent. More precisely, this risk comes from imperfect predictions of the opponent moves. Both OM search and PrOM search rely on knowing exactly which move the opponent is going to take. When the opponent deviates from the predicted move, the result might be disastrous for the player. Imperfect predictions can come from imperfect knowledge, but also from normal search itself. This is because when both players are using the same search depth, say 6 ply, the opponent's moves are predicted by using only a 5-ply search. The one-ply difference between the prediction and the actual search can cause deviations from the predicted move.

In chapter 4 we showed that there is another risk involved in using opponent models in search, namely the risk caused by overestimations in the own evaluation function. The risk is especially present if the overestimation of a position is large and the opponent does not overestimate that position too. Such an overestimated position can act as an attractor: the player is so eager to reach that position that she is willing to speculate on the opponent's moves in order to lure the opponent in the direction of the position. The opponent is willingly following, because he evaluates that position much more profitable than the speculating player. Since the opponent does not make the same estimation error, the outcome once that position is reached, is better for the opponent than for the player. We stated a condition on the pair of evaluation functions (admissibility) that forbids these situations. In the case when the differences between the evaluation functions are small, the effects of these errors will be limited, too.

From a game-theoretic point of view, the opponent model in OM search is equivalent to a pure strategy. In chapter 6 we introduced PrOM search and showed that the probabilistic opponent model on which it is based is equivalent to a mixed strategy in game theory. The introduction of probabilities in the model gives rise to yet another type of risk: not only the individual opponent types might be wrong, also the probabilities themselves could be inaccurate.

In chapter 6 we hypothesized that the best opponent-probabilities for PrOM search are those that predict the opponent's move the best. However, the experiments in chapter 7 showed that this hypothesis is not true.

### 8.1.2 Implementation

Because of the complex game trees involved, purely theoretical research would not be able to provide a full answer to the problem statement. Moreover, we also needed to find out whether the search methods have any practical use. Therefore, the methods had to be implemented into algorithms and the properties of those algorithms had to be investigated. This led to the second research question.

> **Research question 2**: *How can these search methods be implemented efficiently?*

In chapter 3 we presented two basic approaches for implementing OM search: a one-pass approach (OM$^{\beta 1p}$) and an approach that uses- $\alpha$-$\beta$ probes (OM$^{\beta Pb}$), both enhanced with $\beta$ pruning. We showed that OM$^{\beta 1p}$ has a better best-case performance than OM$^{\beta Pb}$ but experiments indicated that OM$^{\beta 1p}$ performs worse than OM$^{\beta Pb}$ in the case of random game trees, especially with increasing search depth. Furthermore, our analysis of standard search enhancements showed that there are more opportunities for these enhancements in OM$^{\beta Pb}$ than there are in OM$^{\beta 1p}$, resulting in an even larger gap between the two approaches.

The best-case analysis and the random game-tree experiments showed that the number of evaluations that OM$^{\beta Pb}$ needs for a given tree divided by the number of evaluations that $\alpha$-$\beta$ search needs, is approximately linear in the search depth and in the branching factor. In terms of asymptotic complexity this means that using OM$^{\beta Pb}$ instead of $\alpha$-$\beta$ search costs only a constant times the costs of one ply of extra search. The one-pass algorithm OM$^{\beta 1p}$ behaved above-linear with respect to the search depth in the random game-tree experiments.

The implementation of PrOM search was subject of chapter 6. Analogous to OM search, we proposed two approaches: a one-pass approach (PrOM$^{\beta 1p}$) and an approach that uses $\alpha$-$\beta$ probes (PrOM$^{\beta Pb}$), both also enhanced with $\beta$ pruning. Pruning is less profound in PrOM search than in OM search since deep pruning is not allowed.

The best-case analysis showed a worse time complexity for PrOM$^{\beta 1p}$ than for PrOM$^{\beta Pb}$, but the number of evaluations that both algorithms needed, divided by the number of $\alpha$-$\beta$ evaluations, seemed to be an exponential function of the search depth (for odd search depths – for even search depths the relation is linear). The analysis on random game trees in chapter 7 indicated that on these trees, both PrOM$^{\beta 1p}$ and PrOM$^{\beta Pb}$ behave exponentially with respect to the search depth, for odd as well as even depths. Like in the best-case analysis, PrOM$^{\beta Pb}$ used less evaluations than PrOM$^{\beta 1p}$ on the random game trees. We also studied the application of standard search enhancements on PrOM search and, again, the probing version offered more opportunities for application than the one-pass version.

So, to recapitulate, in general we may state that the more efficient way to implement OM search and PrOM search is by using $\alpha$-$\beta$ probes, preferably enhanced with transposition tables. However, the computational costs of PrOM search are considerable, especially for larger search depths.

### 8.1.3 Practical applicability

Now that efficient implementations of the search methods were possible, these algorithms had to be tested in a realistic setting because the ultimate reply to our problem statement depends on whether and when these algorithms work in practice.

> **Research question 3**: *What is the nature of the practical circumstances in which OM search and PrOM search perform better than the current implementations of Minimax search?*

We tested the search algorithms in three real board games: in lines of action (LOA), in the chess endgame KQKR, and in bao. Furthermore, Carmel and Markovitch (1998) had performed experiments in checkers and Gao *et al.* (2001) performed some experiments in the game of othello. We discuss those experiments along with our own ones. We will treat the experiments in the following order: LOA, chess, checkers, othello, and finally bao.

#### LOA experiments

The experiments in LOA were disastrous for OM search (see table 5.1). Even with perfect knowledge of the opponent's evaluation functions, OM search played considerably worse than Minimax search in most cases. In these experiments, the search depth was kept constant, which should provide an advantage to the OM-search player. The bad results for OM search can probably be ascribed to the poor quality of the evaluation functions and to the absence of admissibility.

The experiments with PrOM search were more fortunate (see tables 7.1, 7.2, and 7.3). In two of the three tournaments, PrOM search outperformed Minimax search. Also in these experiments, the search depth was kept constant, which meant an even bigger advantage for PrOM search.

Since the number of games was fairly small in the LOA experiments (60 per setting), it is not possible to formulate detailed conclusions from them. Furthermore, the search depth was shallow (2 to 5 plies) and the test set of 30 positions used might have been not too representative. Together this means that the LOA experiments provided only a preliminary judgement on the practical applicability of OM search and PrOM search.

#### KQKR experiments

The OM-search experiments in the KQKR chess endgame (section 5.4) were conducted in order to test the performance of OM search in a situation in which admissibility was guaranteed by using an endgame database. Furthermore, the OM search player was provided with perfect knowledge of the opponent's evaluation function. In these experiments we also tested our hypothesis on the effect of the speculative search depth.

The results of this experiment were not conclusive with respect to OM search. In the first one of the two positions used, OM search most of the times performed better than Minimax search, but in the second position, OM search performed most of the times worse. Since the number of games per setting was 20, the outcomes were statistically significant only in a few cases.

Further analysis of the games played revealed a difference in the two positions that could explain the difference in performance of OM search. Figure 5.13 revealed that games played from the second position showed less diversity than games played from the first position. This could have led to less opportunities for the OM search player to lure the opponent into unprofitable situations.

The KQKR experiments showed that perfect knowledge of the opponent's evaluation function, a perfect evaluation function (i.e., an endgame database), and admissibility together are still not sufficient to make OM search successful.

We did not find sufficient evidence to reject our hypothesis on the effect of the speculative search depth. A statistical test (Anova) showed that the speculative search depth did have a significant influence on the performance, but the effect was not univocal.

### Checkers experiments

Carmel and Markovitch (1998) described experiments with OM search in the game of checkers (see subsection 2.4.4). These experiments were conducted with M* search, but since the applied opponent model had depth 1, the algorithm was equivalent to OM search. The evaluation functions $V_0$ and $V_{op}$ were closely related (see our discussion in section 2.5) and probably were close to admissibility. In contrast to all other experiments discussed in this subsection, the players in these experiments had only limited resources: the total number of evaluations per move was restricted.

In this setting, the OM-search implementation appeared to outperform $\alpha$-$\beta$ search. The improvement is not very large, but statistically significant. The experiments showed a remarkable effect: the improvement diminished when more resources became available to the players.

### Othello experiments

The experiments in othello with $(D, d)$-OM search by Gao *et al.* (2001) (see subsection 2.4.3), showed a slight improvement of the performance if $(D, d)$-OM search was used instead of Minimax. The improvement seemed to increase with larger difference in search depth between the players. (It should be noted, however, that because of a small sample size, none of the measured improvements were statistically significant.) In these experiments, the speculating player was provided with perfect knowledge of the opponent's evaluation function since both players used the same one. The true opponent's search depth was also known. Furthermore, the speculating player was given an additional advantage because of the absence of time restrictions.

The experiment indicated that when playing against an opponent using the same evaluation function as you, but searching less deep than yourself, $(D, d)$-OM search might be profitable.

### Bao experiments

We performed two unconnected experiments in bao. The bao experiments with OM search in section 5.5 were designed to find out how OM search can be brought to good results. In these experiments, the opponent had perfect knowledge of the opponent's evaluation function, no time restriction was given and the speculative search depth of OM search was limited to 1.

It appeared that a combination of good opponent prediction and extended search depth was needed for good results. Of these two factors, the extended search depth seemed to be more important than the good prediction. The quality of the evaluation functions appeared to be important for the effect of OM search. Because the evaluation functions did not obey the admissibility demand, the results for plain OM search were not good for most of the players. With additional resources, however, OM search was made successful. The last tournament (see table 5.11) showed that strict risk avoidance also led to good performance.

The experiments with PrOM search in section 7.5 differed in one important aspect from the OM-search experiments: the player using PrOM search did not have perfect knowledge of the opponent. The speculating player was still given advantages since the search time was unrestricted. OM search was also included in the PrOM search experiments, since one of the probabilistic opponent models was equivalent to OM search.

In these experiments PrOM search always performed better than OM search (as soon as $\Pr(\omega_0)$ was larger than 0). It also performed better than Minimax search in some cases. In those cases, the improvement was small but significant.

### Recapitulation

The experiments showed that it is not a straightforward matter to apply OM search successfully in a practical setting. There are several factors that influence the performance. Below we mention nine of them.

- the extent of the knowledge of the opponent's evaluation function (the more knowledge the better)
- the extent of the knowledge of the opponent's search depth (the more knowledge the better)
- the quality of both evaluation functions (the higher quality the better)
- the difference in quality of the evaluation functions (the larger difference the better)
- the existence of admissibility of the pair of evaluation functions (admissibility is a pre)
- the quality of the prediction of the opponent (the higher quality the better)
- the quality of the prediction of the own moves (the higher quality the better)

- the (effective) branching factor (the larger the better)
- the search depth (ambiguous)

Since the performance of OM search varied from game to game, there must exist other, game-dependent, factors that influence the performance.

Whatever the case, PrOM search seems to have more potential than OM search. However, the computational costs of the former are much higher. The results of PrOM search for LOA were better than those for bao, but in LOA, one of the opponent types was the same as the evaluation function that the opponent really used, whereas in bao, the opponent used an other evaluation function than any of the opponent types.

### 8.1.4 Learning opponent models

Since PrOM search theoretically has the possibility to be used when learning opponent models, we wished to test whether this is possible in a practical setting.

> **Research question 4**: *Can probabilistic Opponent-Model search be used in practice to learn an opponent model?*

The answer to this research question is negative. The bao experiments in section 7.5, and especially the results in figure 7.17, show that the opponent-type probabilities that predict the opponent's moves the best are not the opponent-type probabilities that lead to the best performance of PrOM search. This means that on-line learning based on the move probabilities of PrOM search as described in subsection 6.6.2 is not feasible. To find optimal opponent-type probabilities, other techniques will be needed, such as reinforcement learning. It is questionable if such a technique can be used for on-line learning.

## 8.2 Answer to the Problem Statement

Our problem statement was:

> **Problem statement**: *Under what conditions can Opponent-Model search and Probabilistic Opponent-Model search improve computer game-playing?*

Taking the answers to the research questions above into consideration, the answer to the problem statement must be that OM search or PrOM search can improve computer game-playing, but only when the circumstances are sufficiently profitable. Both search methods need (far) more resources than the current search methods based on $\alpha$-$\beta$ search. Moreover, especially OM search suffers from many risks. If both methods are applied without serious precautions, the performance of computer game-playing is bound to decrease instead of to improve.

## 8.3 Contributions and Significance

In this section we list the contributions of the thesis and state its significance. There are 3 main contributions and 3 additional contributions.

### Main contributions

The first main contribution of this thesis is the increased knowledge about using opponent models in computer game-playing. We made clear that more risk is involved in using opponent models by means of OM search (and its derivatives) than was anticipated so far. This risk does not only come from imperfect knowledge of the opponent, or from failures to predict the opponent's moves precisely, it also comes from errors in the own evaluation function. This risk-sensitiveness of OM search (and PrOM search) reveals an important property of Minimax search. Apparently, Minimax search can render estimation errors in an evaluation function harmless; a property that outweighs the lack of an opponent model in Minimax.

The second main contribution is a search method that is based on a probabilistic opponent model: PrOM search. This method is an extension of the existing OM search. The method is designed to incorporate a player's uncertainty of the opponent. It appears to have a built-in risk management.

The third main contribution consists a number of algorithms for both OM search and PrOM search. The algorithms can be divided into one-pass algorithms and algorithms that use $\alpha$-$\beta$ probes. For the four most important algorithms, best-case analyses and average-case (random-game) analyses were provided. We also showed the applicability of standard search enhancements, such as transposition tables.

### Additional contributions

Our first additional contribution is an analysis of the ordering of evaluation functions in chapter 4. We showed that the simple ordering relation 'better' between evaluation function does not exist. There are at least eight different ordering relations possible; each of them has its own particular use in a specific context.

The second additional contribution of the thesis is the computer-science introduction to the fairly unknown game of bao, which has some interesting properties. The game is regarded as the most complex member of the mancala family. Some mancala games have already been solved (e.g., awari and kalah), but bao differs so much from these games that solving bao is a challenge on its own. Furthermore, no competition in computerized bao has been held yet.

The third additional contribution is partly bridging the gap that has grown between computer game-playing and game theory. Our attempt was twofold: (1) using the language of game theory in our analysis and (2) showing that some subjects in computer game-playing have parallels in modern game theory. In particular we mention here the relation between some of the search methods discussed in chapter 2 and the recent discussion on subgame-perfect equilibria in game theory. Another issue is the importance of nonzero-sum games in computer game-playing since heuristic evaluation functions sometimes have asymmetric aspects that destroy the zero-sum character of the game.

**Significance**

The significance of the thesis is that it makes clear that using opponent models in game-tree search can be profitable, but that the circumstances have to be ideal and that serious precautions have to be taken.

## 8.4 Recommendations for Future Research

We end the chapter by listing nine recommendations for future research.

1. **Further study of risk management in OM search and PrOM search.** In our experiments we only applied risk avoidance with OM search in the KQKR experiments and in one of the bao tournaments. In particular the results of the seventh bao tournament show that risk management might be a promising way to improve OM search, but maybe also PrOM search.

2. **Application to other game domains.** The performance of OM search and PrOM search seem to be game-dependent. It could therefore be the case that there are game domains in which these methods perform sufficiently well to be of practical interest. A game domain is to be understood as a specific game or as a specific phase of a game.

3. **Application of reinforcement learning for obtaining optimal opponent-type probabilities.** Since our hypothesis on the opponent-type probabilities had to be rejected, further research is needed to find means for obtaining these probabilities. One option might be reinforcement learning.

4. **Combination of PrOM search and $(D, d)$-OM search.** An interesting specialisation of PrOM search might be the following: instead of providing a separate evaluation function per opponent type, all opponent types use the same evaluation function, but each one uses a different search depth. It could be that such an application of PrOM search suffers from less risk than OM search and could handle uncertainty about the opponent's true search depth.

5. **Parallel computing.** The implementations of OM search and PrOM search could benefit from parallel computing. For instance, the execution of independent $\alpha$-$\beta$ probes can be done in parallel. In the case of PrOM search, the $\alpha$-$\beta$ probes for each opponent type at a node could be delegated to separate processors.

6. **Further study of bounded-sum pruning.** The case in which evaluation functions are closely related by a bounded sum is profitable to OM search since the risk of overestimations is inherently small. Furthermore, pruning can be more effective in this case. More studies in different game domains are needed to find out whether this has a practical use.

7. **Search in near-zero-sum games.** Closely related to the previous recommendation is a study of games that are not zero-sum, or of which the reduced

games are not zero-sum, but of which the differences of the payoffs are small. New search methods have to be developed for this case and an analysis of risks and gains has to be performed.

8. **New equilibria in game-tree search.** As indicated at the end of chapter 2, a study of the applicability of new sorts of equilibria from game theory in computer game-tree search might lead to improved search algorithms.

9. **A further study of bao.** The game of bao has many strategies and tactics that have not been incorporated in a computer program yet. Although the game is easily won by a computer against a human (Donkers, 2002), a true computer competition would bring new knowledge of the game and its players. Furthermore, since bao is much more complex than awari and kalah, solving the game poses a new challenge to computer science.

*Why did my parents send me to the schooles*
*That I with knowledge might enrich my mind?*
*Since the desire to know first made men fools,*
*And did corrupt the root of all mankind.*

...

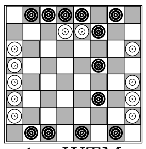
*My selfe am center of my circling thought,*
*Only my selfe I studie, learne, and know.*

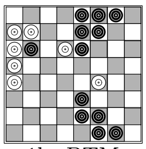
*Sir John Davies - Nosce Teipsum (1599)*

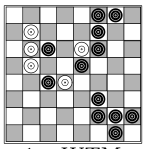
# Appendix A

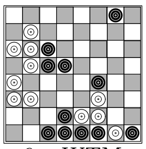
# Loa Test Set

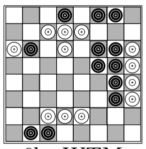
Below we reproduce the set of positions from the game of loa as used in the experiments. From 10 games, three positions were selected, one at the beginning of the game, one at the middle, and one at the end. Below every position it is indicated whether black (BTM) or white (WTM) is to move.

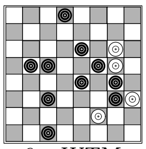

1a: WTM

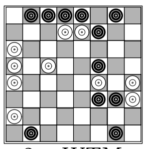

1b: BTM

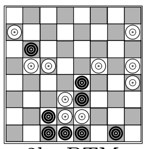

1c: WTM

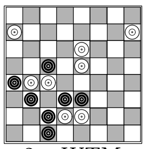

6a: WTM

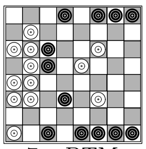

6b: WTM

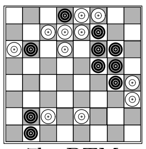

6c: WTM

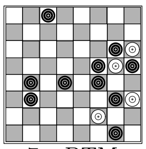

2a: WTM

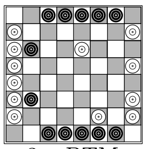

2b: BTM

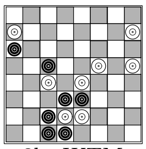

2c: WTM

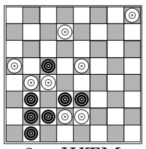

7a: BTM

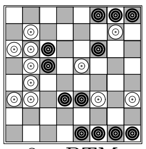

7b: BTM

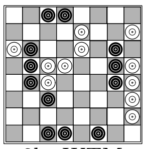

7c: BTM

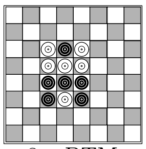

3a: BTM

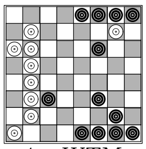

3b: WTM

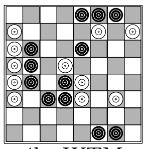

3c: WTM

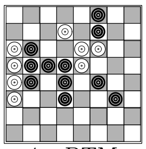

8a: BTM

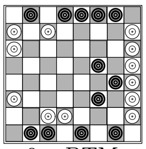

8b: WTM

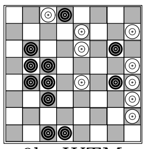

8c: BTM

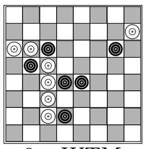

4a: WTM

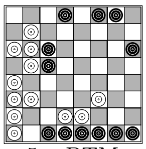

4b: WTM

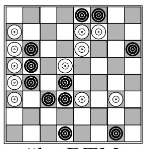

4c: BTM

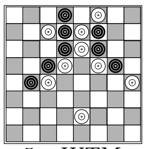

9a: BTM

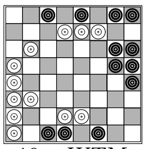

9b: WTM

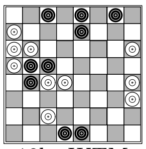

9c: WTM

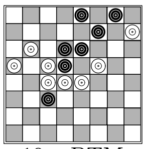

5a: BTM


5b: BTM


5c: WTM


10a: WTM


10b: WTM

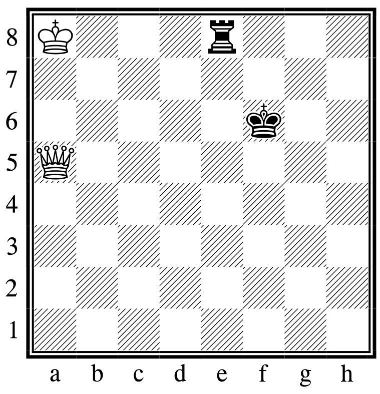

10c: BTM

# Appendix B

# Browne-Belle Games

Below we present the two games that Walter Browne played against Ken Thompson's chess machine Belle in 1977. Annotated versions of these games can be found in Newborn (1997). A thorough analysis of both games is provided in Haworth (2003). Next to the games we provide two versions of optimal play for each starting position. One version is generated by the program Chess Genius Classic using Ken Thompson's distance to conversion (DTC) endgame databases. The other version is generated by the program Crafty using Eugene Nalimov's distance to win (DTW) endgame databases. Both positions are won in 35 moves if both sides play optimally, regardless whether the DTC or the DTW database is used.

## First Game

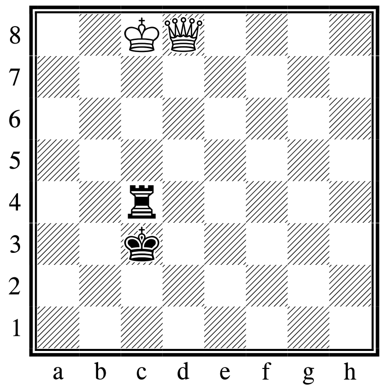


**White: Walter Browne, Black: Belle (1977, New York).** 1. Kb7 Re7+ 2. Kc6 Re6+ 3. Kd7 Re7+ 4. Kd8 Re4 5. Qc5 Re5 6. Qd4 Kf5 7. Kd7 Re4 8. Qd3 Kf4 9. Kd6 Re3 10. Qd4+ Re4 11. Qf2+ Kg4 12. Kd5 Re8 13. Qf6 Re3 14. Kd4 Rf3 15. Qg6+ Kf4 16. Qg2 Ra3 17. Qc6 Ra1 18. Qc7+ Kf5 19. Qc2+ Ke6 20. Qd2 Ra7 21. Qb4 Re7 22. Ke4 Kf6+ 23. Kf4 Ke6 24. Qd4 Rf7+ 25. Ke4 Rf6 26. Qd5+ Ke7 27. Ke5 Rh6 28. Qb7+ Kd8 29. Qf7 Rc6 30. Kd5 Rb6 31. Kc5 Ra6 32. Qc4 Rf6 33. Qh4 Ke7 34. Kd5 Kf7 35. Ke5 Re6+ 36. Kf5 Rd6 37. Qc4+ Ke7 38. Ke5 Rh6 39. Qc7+ Kf8 40. Kf5 Ke8 41. Qc1 Rd6 42. Qc8+ Ke7 43. Qc7+ Rd7 44. Qc5+ Kd8 45. Ke6 Rb7 $^1/_2$ – $^1/_2$ (draw agreed)

**Optimal play according to Thompson's DTC tables** (Generated using Chess Genius Classic). 1. Ka7 Re7+ 2. Kb6 Re6+ 3. Kc7 Re7+ 4. Kd8 Re4 5. Qc5 Re5 6. Qc3 Ke6 7. Kc7 Rd5 8. Qe3+ Re5 9. Qb3+ Kf5 10. Qf3+ Ke6 11. Kc6 Ke7 12. Qg3 Kf6 13. Kd6 Re6+ 14. Kd5 Kf7 15. Qc7+ Re7 16. Qf4+ Kg6 17. Qg4+ Kf6 18. Kd6 Re8 19. Qf4+ Kg6 20. Kd7 Rg8 21. Ke7 Kh5 22. Qh2+ Kg6 23. Qh4 Rg7+ 24. Ke6 Ra7 25. Qe4+ Kh5 26. Qh1+ Kg5 27. Qg1+ Kf4 28. Q×a7 Ke4 29. Qb6 Kd3 30. Kd5 Kc3 31. Qd4+ Kc2 32. Kc4 Kb1 33. Kc3 Ka1 34. Qb4 Ka2 35. Qb2+ 1–0.

**Optimal play according to Nalimov's DTW tables** (Generated using CRAFTY). 1. Kb7 Re7+ 2. Kc8 Re8+ 3. Kd7 Re7+ 4. Kd8 Re3 5. Qb6+ Re6 6. Qb2+ Kf5 7. Kd7 Re3 8. Kd6 Ke4 9. Ke6 Kf4+ 10. Kd5 Rd3+ 11. Kc4 Re3 12. Qf6+ Kg4 13. Kd4 Re2 14. Qg6+ Kf4 15. Qh6+ Kg4 16. Qf6 Re1 17. Kd3 Kg3 18. Qg5+ Kf3 19. Qf5+ Kg3 20. Kd2 Rg1 21. Ke2 Rg2+ 22. Kf1 Rh2 23. Qe5+ Kh3 24. Qd6 Rh1+ 25. Kf2 Kg4 26. Qd4+ Kf5 27. Qd5+ Kf6 28. Q×h1 Ke7 29. Qc6 Kf7 30. Kg3 Ke7 31. Kg4 Kf7 32. Kf5 Ke7 33. Qc7+ Ke8 34. Kg6 Kf8 35. Qd8 1–0.

## Second Game



**White: Walter Browne, Black:** BELLE **(1977, New York).** 1. Kb7 Rb4+ 2. Kc6 Rc4+ 3. Kb5 Rb4+ 4. Ka5 Re4 5. Qd6 Rd4 6. Qe5 Kd3 7. Kb5 Re4 8. Qf6 Ke3 9. Kc5 Rf4 10. Qg6 Ra4 11. Qg3+ Ke2 12. Qc3 Rf4 13. Kd5 Rh4 14. Qc2+ Ke3 15. Qd1 Kf2 16. Qd2+ Kf3 17. Qe1 Rg4 18. Qd1+ Kf4 19. Qe2 Rg5+ 20. Kd4 Rf5 21. Qe3+ Kg4 22. Ke4 Rf7 23. Qg1+ Kh5 24. Qg3 Rf8 25. Ke5 Rf7 26. Ke6 Rf8 27. Qa3 Rf4 28. Qh3+ Kg5 29. Qg3+ Rg4 30. Qe5+ Kh4 31. Qh2+ Kg5 32. Ke5 Kg6 33. Qh8 Rg5+ 34. Ke6 Rg4 35. Qg8+ Kh5 36. Qh7+ Kg5 37. Ke5 Rg3 38. Qg7+ Kh4 39. Qh6+ Kg4 40. Ke4 Rg2 41. Qg6+ Kh3 42. Qh5+ Kg3 43. Ke3 Rg1 44. Qg5+ Kh2 45. Qh4+ Kg2 46. Ke2 Ra1 47. Qe4+ Kh3 48. Qh7+ Kg3 49. Qg7+ Kh3 50. Q×a1 1–0 (Black resigns)

**Optimal play according to Thompson's DTC tables** (Generated using CHESS GENIUS CLASSIC). 1. Kb8 Rb4+ 2. Ka7 Ra4+ 3. Kb6 Rb4+ 4. Ka5 Re4 5. Qd6 Rd4 6. Qa3+ Kc4 7. Kb6 Rd2 8. Qf8 Rd4 9. Qf7+ Kd3 10. Qf3+ Kc4 11. Kc6 Kb4 12. Qf2 Kc3 13. Kc5 Rc4+ 14. Kd5 Kb3 15. Qe2 Rc2 16. Qd3+ Rc3 17. Qb5+ Ka3 18. Kd4 Rg3 19. Qb6 Rb3 20. Qa5+ Kb2 21. Kc4 Ra3 22. Qb4+ Ka2 23. Qb8 Rh3 24. Qg8 Rb3 25. Qd5 Rb2 26. Kc3+ Kb1 27. Qh1+ Ka2 28. Qd1 Rh2 29. Qa4+ Kb1 30. Qe4+ Ka1 31. Qa8+ Kb1 32. Qb8+ Ka2 33. Q×h2+ Ka3 34. Kc4 Ka4 35. Qa2+ 1–0.

**Optimal play according to Nalimov's DTW tables** (Generated using CRAFTY). 1. Kb7 Rb4+ 2. Kc6 Rc4+ 3. Kb5 Rb4+ 4. Ka5 Re4 5. Qd6 Rd4 6. Qf6 Kd3 7. Kb5 Ke3 8. Kc5 Rf4 9. Qe5+ Re4 10. Qc3+ Kf4 11. Kd5 Re2 12. Qf6+ Kg3 13. Qf5 Re1 14. Qh5 Re3 15. Kd4 Ra3 16. Qd5 Ra1 17. Ke3 Re1+ 18. Kd2 Rf1 19. Qg5+ Kf3 20. Qf5+ Kg2 21. Qg4+ Kh2 22. Ke3 Rg1 23. Qh4+ Kg2 24. Ke2 Ra1 25. Qe4+ Kg3 26. Qe5+ Kg4 27. Q×a1 Kf5 28. Kf3 Ke6 29. Kf4 Kd5 30. Qa5+ Kc4 31. Ke4 Kb3 32. Kd3 Kb2 33. Qb4+ Ka2 34. Kc3 Ka1 35. Qb2 1–0.

# Appendix C

# Zanzibar Bao Rules for the Computer

In this appendix we provide a version of the rules for Zanzibar bao especially adapted for computer use. These rules are based on De Voogt's (1995) thesis.

## The Board

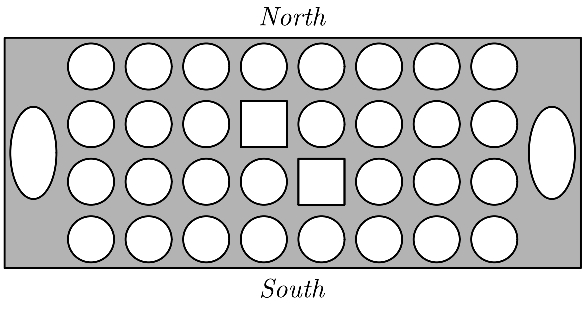


Figure C.1: The bao board.

Zanzibar bao (or bao for short) is a game from the mancala family (De Voogt, 1995). It is played by two persons on a board with four rows of eight pits or holes, see figure C.1. 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.

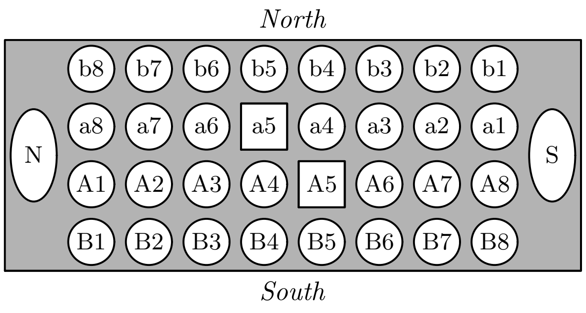


Figure C.2: Numbering of the holes in bao.

The holes on each rows are numbered from 1 to 8, starting at the left hand of the player owning the rows, see figure C.2. 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 figure C.2. 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 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 figure C.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.

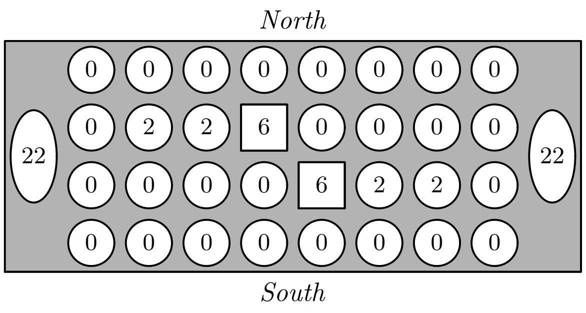


Figure C.3: Official opening position of bao.

**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).

**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 figure C.4.

```
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
```

Figure C.4: A bao game transcript.

# Appendix D

# Bao Test Set

The following table gives the 100 start positions used in the bao experiments of section 5.5. 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 figure C.2. 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 |

| 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 |
| 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 |

| 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 |
| 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

Akl, S.G. and Newborn, M.M. (1977). The Principle Continuation and the Killer Heuristic. *ACM Annual Conference*, pp. 466–473. [56]

Allis, L.V., Meulen, M. van der, and Herik, H.J. van den (1991). Databases in Awari. *Heuristic Programming in Artificial Intelligence 2: The Second Computer Olympiad* (eds. D.N.L. Levy and D.F Beal), pp. 73–86, Ellis Horwood, Chichester, UK. ISBN 0–13–382615–5. [94]

Allis, L.V., Meulen, M. van der, and Herik, H.J. van den (1994). Proof-Number Search. *Artificial Intelligence*, Vol. 66, No. 1, pp. 91–124. [8]

Anantharaman, T. (1997). Evaluation Tuning for Computer Chess: Linear Discriminant Methods. *ICCA Journal*, Vol. 20, No. 4, pp. 224–242. [125]

Anantharaman, T., Campbell, M., and Hsu, F.H. (1988). Singular Extensions: Adding Selectivity to brute-Force Searching. *AAAI Spring Symposium*, pp. 8–13. [58]

Aumann, R.J. (1995). Backward Induction and Common Knowledge of Rationality. *Games and Economic Behavior*, Vol. 8, pp. 6–19. [34]

Ballard, B.W. (1983). The $*$-Minimax Search Procedure for Trees Containing Chance Nodes. *Artificial Intelligence*, Vol. 21, No. 3, pp. 327–350. [17]

Baudet, G. (1978). *The Design and Analysis of Algorithms for Asynchronous Multiprocessors.* Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University. [59]

Baxter, J., Trigdell, A., and Weaver, L. (1998). KnightCap: a Chess Program that Learns by Combining TD($\lambda$) with Game-Tree Search. *Proc. 15th International Conf. on Machine Learning*, pp. 28–36, Morgan Kaufmann, San Francisco, CA. [62, 96]

Beal, D.F. (1989). Experiments with the Null Move. *Advances in Computer Chess V* (ed. D.F. Beal), pp. 65–79, Pergamon Press, Oxford, England. ISBN 0–444–87159–4. [58]

Bell, A.G. (1968). Kalah on Atlas. *Machine Intelligence*, Vol. 3, pp. 181–194. [93]

Berliner, H. (1977). Search and Knowledge. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-77)*, pp. 975–979. [32]

Billings, D., Davidson, A., Schaeffer, J., and Szafron, S. (2000). The Challenge of Poker. *Artificial Intelligence*, Vol. 134, Nos. 1–2, pp. 201–240. [126]

Binmore, K. (1996). A Note on Backward Induction. *Games and Economic Behavior*, Vol. 17, pp. 135–137. [34]

Björnsson, Y.L. (2002). *Selective Depth-First Game-Tree Search.* Ph.D. thesis, University of Alberta. [59]

Breuker, D.M. (1998). *Memory versus Search in Games.* Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands. ISBN 90–9012006–8. [57]

Breuker, D.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1994). Replacement Schemes for Transposition Tables. *ICCA Journal*, Vol. 17, No. 4, pp. 183–193. [57, 79]

Brudno, A.L. (1963). Bounds and Variations for Abridging the Search of Estimates. *Problems of Cybernetics*, Vol. 10, pp. 225–241. Translation of Russian original in *Problemi Kibernetiki*, Vol. 10, May 1963, pp. 71–76. [59]

Bruin, A. de, Pijls, W., and Plaat, A. (1994). Solution Trees as a Basis for Game Tree Search. *ICCA Journal*, Vol. 17, No. 4, pp. 207–219. [6]

Carmel, D. and Markovitch, S. (1993). Learning Models of Opponent's Strategies in Game Playing. *Proceedings AAAI Fall Symposion on Games: Planning and Learning*, pp. 140–147, Raleigh, NC. [10, 24, 25, 27, 28, 30, 108, 150]

Carmel, D. and Markovitch, S. (1994). The $M^*$ Algorithm: Incorporating Opponent Models in Adversary Search. Technical Report CIS9402, Technion, Haifa, Israel. [30]

Carmel, D. and Markovitch, S. (1996a). Learning and Using Opponent Models in Adversary Search. Technical Report CIS9609, Technion, Haifa, Israel. [24, 27, 28, 30, 31, 32, 33, 35, 40, 47, 117]

Carmel, D. and Markovitch, S. (1996b). Incorporating Opponent Models into Adversary Search. *Proceedings AAAI-96*, pp. 120–125, Portland, OR. [30]

Carmel, D. and Markovitch, S. (1998). Pruning Algorithms for Multi-Model Adversary Search. *Artificial Intelligence*, Vol. 99, No. 2, pp. 325–355. [30, 31, 40, 47, 48, 118, 152, 153]

Carmel, D. and Markovitch, S. (1999). Exploration Strategies for Model-based Learning. *Autonomous Agents and Multi-Agent Systems*, Vol. 2, No. 2, pp. 141–272. [106, 126]

Christensen, J. and Korf, R.E. (1986). A Unified Theory of Heuristic Evaluation Functions and its Application to Learning. *Proceedings AAAI-86*, pp. 148–152, Portland, OR. [63, 64]

Davison, B.D. and Hirsh, H. (1998). Predicting Sequences of User Actions. *Predicting the Future: AI Approaches to Time-Series Problems*, pp. 5–12, AAAI Press, Madison, WI. Proceedings of AAAI-98/ICML-98 Workshop, published as Technical Report WS-98-07. [124]

Domingos, P. and Pazzani, M. (1997). On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, Vol. 29, pp. 103–130. [124]

Donkers, H.H.L.M. (2002). The Game of Bao - Report from the 7th Computer Olympiad. *ICGA Journal*, Vol. 25, No. 4, p. 255. [158]

Donkers, H.H.L.M. and Uiterwijk, J.W.H.M. (2002). Programmming Bao. *Seventh Computer Olympiad: Computer-Games Workshop Proceedings* (ed. J.W.H.M. Uiterwijk), Vol. Technical Reports in Computer Science, CS 02-03, IKAT, Department of Computer Science, Universiteit Maastricht, Maastricht. [94]

Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2001). Probabilistic Opponent-Model Search. *Information Sciences*, Vol. 135, Nos. 3–4, pp. 123–149. [47, 106, 117]

Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Voogt, A.J. de (2002). Mancala Games – Topics in Artificial Intelligence and Mathematics. *Step by Step. Proceedings of the 4th Colloquium 'Board Games in Academia'* (eds. J. Retschitzki and R. Haddad-Zubel), Editions Universitaires, Fribourg, Switserland. [94]

Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2003). Admissibility in Opponent-Model Search. *Information Sciences*, Vol. 154, Nos. 3–4, pp. 119–140. [66]

Egnor, D. (2000). Iocaine Powder. *ICGA Journal*, Vol. 23, No. 1, pp. 33–35. [126]

Fenner, C.J. (1979). Computer Chess, News about the North American Computer Chess Championship. *The British Chess Magazine*, Vol. 99, No. 5, pp. 193–200. [84]

Fudenberg, D. and Levine, D.K. (1998). *The Theory of Learning in Games*. MIT Press, Cambridge, MA. ISBN 0–2620–6194–5. [126]

Fudenberg, D. and Tirole, J. (1991). *Game Theory*. MIT Press, Cambridge, MA. ISBN 0–262–061414–4. [3, 4, 19, 32, 35]

Fürnkranz, J. (1996). Machine Learning in Computer Chess: The Next Generation. *ICCA Journal*, Vol. 19, No. 3, pp. 147–161. [126]

Gao, X., Iida, H., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1999). A Speculative Strategy. *Computers and Games, Lecture Notes in Computer Science LNCS 1558* (eds. H.J. van den Herik and H. Iida), pp. 74–92, Springer Verlag, Berlin, Germany. ISBN 3–540–65766–5. [29]

Gao, X., Iida, H., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2001). Strategies Anticipating a Difference in Search Depth using Opponent-Model Search. *Theorerical Computer Science*, Vol. 252, Nos. 1–2, pp. 83–104. [29, 30, 31, 35, 152, 153]

Goot, R. van der (2001). Awari Retrograde Analysis. *Computers and Games – Second International Conference, CG2000* (eds. T.A. Marsland and I. Frank), Lecture Notes in Computer Science LNCS 2063, pp. 87–95, Springer Verlag, Berlin, Germany. ISBN 3–540–43080–6. [94]

Greenblatt, R.D., Eastlake, D.E., and Crocker, S.D. (1967). The Greenblatt Chess Program. *Proceedings of AFIPS Fall Joint Computing Conference*, Vol. 31, pp. 801–810, Springer Verlag, San Francisco. Reprinted (1988) in Computer Chess Compendium (ed. D.N.L. Levy), pp. 55–66. B.T. Batsford Ltd., London, United Kingdom. [57]

Haworth, G.M.C. (2003). Reference Fallible Endgame Play. *ICCA Journal*, Vol. 26, No. 2. (in press). [161]

Herik, H.J. van den (1983). *Computerschaak, Schaakwereld en Kunstmatige Intelligentie*. Ph.D. thesis, Academic Service, The Hague, The Netherlands. ISBN 90–6233–103–3. [2, 32]

Herik, H.J. van den and Herschberg, I.S. (1985). The Construction of an Omniscient Endgame Database. *ICCA Journal*, Vol. 8, No. 3, pp. 141–149. [58]

Herik, H.J. van den, Uiterwijk, J.W.H.M., and Rijswijck, J. van (2002). Games Solved, Now and in the Future. *AI Journal*, Vol. 134, Nos. 1–2, pp. 277–311. [95]

Holland, J.H. (1975). *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI. ISBN 0–262–58111–6. [95]

Hsu, F.-h. (2002). *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press. ISBN 0–691–09065–3. [1, 56]

Huizinga, J. (1938). *Homo Ludens: Proeve eener Bepaling van het Spel-Element der Cultuur*. Tjeenk Willink, Haarlem, The Netherlands. [2]

Hyatt, R. (2002a). Personal Communication. [86]

Hyatt, R. (2002b). CRAFTY FTP site. ftp://ftp.cis.uab.edu/hyatt. [85]

Iida, H., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1993a). Opponent-Model Search. Technical Report CS 93-03, Universiteit Maastricht, Maastricht, The Netherlands. [24, 25, 27, 29, 43, 150]

Iida, H., Uiterwijk, J.W.H.M., Herik, H.J. van den, and Herschberg, I.S. (1993b). Potential Applications of Opponent-Model Search. Part 1: the domain of applicability. *ICCA Journal*, Vol. 16, No. 4, pp. 201–208. [10, 28, 29, 31]

Iida, H., Uiterwijk, J.W.H.M., Herik, H.J. van den, and Herschberg, I.S. (1994). Potential Applications of Opponent-Model Search. Part 2: risks and strategies. *ICCA Journal*, Vol. 17, No. 1, pp. 10–14. [27, 29]

Iida, H., Handa, K.-i., and Uiterwijk, J.W.H.M. (1995). Tutoring Strategies in Game-Tree Search. *ICCA Journal*, Vol. 18, No. 4, pp. 191–204. [28]

Iida, H., Kotani, I., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1997a). Gains and Risks of OM Search. *Advances in Computer Chess 8* (eds. H.J. van den Herik and J.W.H.M. Uiterwijk), pp. 153–165, Universiteit Maastricht, Maastricht, The Netherlands. ISBN 9–062–16234–7. [29]

Iida, H., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1997b). Cooperative Strategies for Pair Playing. *Computer Software*, Vol. 14, No. 1, pp. 45–49. [28]

Irving, G., Donkers, H.H.L.M., and Uiterwijk, J.W.H.M. (2000). Solving Kalah. *ICGA Journal*, Vol. 23, No. 3, pp. 139–148. [93]

Jansen, P.J. (1992a). *Using Knowledge About the Opponent in Game-Tree Search*. Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA. [18, 19, 20, 21, 22, 23, 28, 106, 108]

Jansen, P.J. (1992b). KQKR: Awareness of a Fallible Opponent. *ICCA Journal*, Vol. 15, No. 3, pp. 111–131. [18, 84, 88]

Jansen, P.J. (1992c). KQKR: Assessing the Utility of Heuristics. *ICCA Journal*, Vol. 15, No. 4, pp. 179–191. [18]

Jansen, P.J. (1993). KQKR: Speculatively Thwarting a Human Opponent. *ICCA Journal*, Vol. 16, No. 1, pp. 3–18. [18, 19, 20, 21, 22, 84, 108]

Johnson, P. and Lancaster, A. (2000). Swarm User Guide. http://www.swarm.org/swarmdocs/userbook/userbook.html. See appendix C. [74]

Junghanns, A. (1998). Are there Practical Alternatives to Alpha-Beta? *ICCA Journal*, Vol. 21, No. 1, pp. 14–32. [2, 8, 10, 19, 20]

Kaindl, H., Horacek, H., and Wagner, M. (1986). Selective Search versus Brute Force. *ICCA Journal*, Vol. 9, No. 3, pp. 140–145. [58]

Kajihara, Y., Hashimoto, T., and Iida, H. (2002). Speculative Play in Shogi Endgame. Technical Report CGRI-014-02, Computer Games Research Institute, Shizuoka University, Hamamatsu, Japan. [21]

KALAH (1963). Kalah Recognised as Valuable Educational Aid. http://www.ahs.uwaterloo.ca/∼museum/countcap/pages/newskala.html. Reprinted from the Melrose Free Press, Dec. 19, 1963. [93]

Knuth, D.E. and Moore, R.W. (1975). An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, Vol. 6, No. 4, pp. 293–326. [8, 43, 50]

Kocsis, L., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2001). Learning Time Allocation using Neural Networks. *Computers and Games – Second International Conference, CG2000* (eds. T.A. Marsland and I. Frank), Lecture Notes in Computer Science LNCS 2063, pp. 170–185, Springer Verlag, Berlin, Germany. ISBN 3–540–43080–6. [82]

L' Ecuyer, P. (2001). Software for Uniform Random Number Generation: Distinguishing the Good and the Bad. *Proceedings of the 2001 Winter Simulation Conference*, pp. 95–105, IEEE Press. [74]

Levy, D.N.L. and Newborn, M.M. (1991). *How Computers Play Chess.* Computer Science Press: W.H. Freeman and Co., New York, NY. ISBN 0–7167–8121–2. [84]

Lincke, T.R. and Marzetta, A. (2000). Large Databases with Limited Memory Space. *ICGA Journal*, Vol. 23, No. 3, pp. 131–138. [94]

Luke, S. (2002). Mersenne Twister in Java. http://cs.gmu.edu/∼sean/research. [74]

Marsland, T.A. (1983). Relative Efficiency of Alpha-Beta Implementations. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-83)*, pp. 763–766. [9, 59]

Matsumoto, M. and Nishimura, T. (1998). Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudorandom Number Generator. *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 1, pp. 3–30. [74]

McAllester, D.A. (1988). Conspiracy Numbers for Min-Max Searching. *Artificial Intelligence*, Vol. 35, No. 3, pp. 827–310. [8]

McCarthy, J. (1990). Chess as the Drosophila of AI. *Computers, Chess, and Cognition* (eds. T.A. Marsland and J. Schaeffer), pp. 227–237, Springer Verlag, Berlin, Germany. Also 'The Fruitfly of AI', *ICCA Journal*, Vol 12, No 4, pp. 199-206. [2]

McCarthy, J. (1997). AI as Sport (Book Review). *Science*, Vol. 276, No. 5216. See also: http://www-formal.stanford.edu/jmc/newborn/newborn.html. [2]

Meulen, M. van der, Allis, L.V., and Herik, H.J. van den (1990). Lithidion, an Awari-Playing Program. Technical Report CS 90-05, Universiteit Maastricht, Maastricht, the Netherlands. [93]

Mood, A.M., Graybill, F.A., and Boes, D.C. (1974). *Introducion to the Theory of Statistics.* McGraw-Hill, Inc., Auckland, New Zealand. ISBN 0–070–85465–3. [123]

Murray, H.J.R. (1952). *A History of Board Games other than Chess.* Oxford University Press, Oxford, UK. ISBN 0878172114. [93]

Nash, J. (1950). Equilibrium Points in $n$-Person Games. *Proceedings of the National Academy of Sciences*, Vol. 36, pp. 48–49. [32]

Neumann, J. von (1928). Zur Theorie der Gesellschaftspiele. *Mathematische Annalen*, Vol. 100, pp. 295–320. [4, 5, 32]

Neumann, J. von and Morgenstern, O. (1944). *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, NJ. [3, 5]

Newborn, M.M. (1977). The Efficiency of the Alpha-Beta Search on Trees with Branch-dependent Terminal Node Scores. *Artificial Intelligence*, Vol. 87, Nos. 1–2, pp. 225–293. [75]

Newborn, M.M. (1997). *Kasparov Versus Deep Blue, Computer Chess Comes of Age.* Springer Verlag, Berlin, Germany. ISBN 0–387–94820–1. [84, 161]

Pazzani, M. and Billsus, D. (1997). Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning*, Vol. 27, pp. 313–331. [124]

Pearl, J. (1980). Scout: A Simple Game-Searching Algorithm with Proven Optimal Properties. *First Annual National Conference on Artificial Intelligence, Stanford.* [59]

Pearl, J. (1984). *Heuristics, Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Company, Reading, MA. ISBN 0–201–05594–5. [62, 69, 71]

Plaat, A. (1996). *Research Re: Search & Re-search.* Ph.D. thesis, Tinbergen Institute and Department of Computer Science, University of Rotterdam, Rotterdam, The Netherlands. [8, 59]

Reibman, A.L. and Ballard, B.W. (1983). Non-Minimax Search Strategies for Use against Fallible Opponents. *AAAI'83*, pp. 338–342, Morgan Kaufmann Publ., San Mateo, CA. [17, 18, 19, 20, 21, 35, 108]

Romein, J.W. and Bal, H.E. (2002). Awari is Solved. *ICGA Journal*, Vol. 25, No. 3, pp. 162–165. [94]

Romein, J.W., Plaat, A., Bal, H.E., and Schaeffer, J. (1999). Transposition Table Driven Work Scheduling in Distributed Search. *16th National Conference on Artificial Intelligence (AAAI'99)*, pp. 725–731. [2]

Russ, L. (2000). *The Complete Mancala Games Book.* Marlow & Company, New York. ISBN 1–56924–683–1. [93]

Russel, R. (1964). Kalah – the Game and the Program. Artificial Intelligence Project Memo nr 22. University of Stanford, Stanford, CA. [93]

Sackson, S. (1992). *A Gamut of Games.* Dover Publications Inc, New York, NY. ISBN 0–486–27347–4. [81]

Samuel, A.L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, Vol. 3, pp. 210–229. [64]

Samuel, A.L. (1967). Some Studies in Machine Learning Using the Game of Checkers II -Recent Progress. *IBM Journal of Research and Development*, Vol. 11, pp. 601–617. [64]

Schaeffer, J. (1983). The History Heuristic. *ICCA Journal*, Vol. 6, No. 3, pp. 16–19. [56]

Schaeffer, J. (1989). The History Heuristic and Alpha-Beta Search Enhancements in Practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. November 1989, pp. 1203–1212. [56]

Schaeffer, J. and Herik, H.J. van den (eds.) (2002). *Chips Challenging Champions. Games, Computers and Artificial Intelligence*, Amsterdam, The Netherlands. Elsevier Science Publishers. ISBN 0–444–50949–6. [2]

Schneider, F. (2001). DEEP JUNIOR wins the 18th World Microcomputer Chess Championship. *ICGA Journal*, Vol. 24, No. 3, pp. 173–179. [86]

Selten, R. (1965). Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit. *Zeitschrift für die gesamte Staatswissenschaft*, Vol. 12, pp. 301–324. [32, 34]

Selten, R. (1975). Re-examination of the Perfectness Concept for Equilibrium Points in Extensive Games. *International Journal of Game Theory*, Vol. 4, pp. 25–55. [34]

Shannon, C.E. (1950). Programming a Computer for Playing Chess. *Philisophical Magazine*, Vol. 7, No. 14, pp. 256–275. [2, 58]

Slagle, J.R. and Dixon, J.K. (1970). Experiments with the M & N Tree-Searching Program. *Communications of the ACM*, Vol. 13, No. 3, pp. 147–154. [10, 16, 17, 21, 35, 93]

Slate, D.J. and Atkin, L.R. (1977). *Chess Skill in Man and Machine*, Chapter Chapter 4. CHESS 4.5 – Northwestern University Chess Program, pp. 82–118. Springer Verlag, Berlin, Germany. [57, 58]

Smith, S.J.J and Nau, D.S. (1994). An Analysis of Forward Pruning. *Proceedings of AAAI 1994*, pp. 1386–1391. [58]

Spearman, C. (1904). The Proof and Measurement of Association between two Things. *American Journal of Psychology*, Vol. 14, pp. 72–101. [65]

Stalnaker, R.C. (1996). Knowledge, Belief and Counterfactual Reasoning in Games. *Economics and Philosophy*, Vol. 12, pp. 133–163. [34]

Stockman, G.C. (1979). A Minimax Algorithm better than Alpha-Beta? *Artificial Intelligence*, Vol. 12, No. 2, pp. 179–196. [8]

Ströhlein, T. (1970). Untersuchungen über Kombinatorische Spiele. M.Sc. thesis, Fakultät für Allgemeine Wissenschaften der Technischen Hochshule München. [58]

Sun (2002). Java$^{TM}$ 2 SDK, Standard Edition Documentation. http://java.sun.com/j2se/1.4/docs. [74]

Sutton, R.S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, Vol. 3, No. 1, pp. 9–44. [82]

Thompson, K. (1986). Retrograde Analysis of Certain Endgames. *ICCA Journal*, Vol. 9, No. 3, pp. 131–139. [18]

Uiterwijk, J.W.H.M. and Herik, H.J. van den (1994). Speculative Play in Computer Chess. *Advances in Computer Chess 7* (eds. H.J. van den Herik, I.S. Herschberg, and J.W.H.M. Uiterwijk), pp. 79–90, Rijksuniversiteit Limburg, Maastricht, The Netherlands. ISBN 9–062–16101–4. [16, 20, 21, 22, 35]

Voogt, A.J. de (1995). *Limits of the Mind. Towards a Characterisation of Bao Mastership*. Ph.D. thesis, University of Leiden, The Netherlands. ISBN 90–73782–50–3. [94, 163]

Wilkins, D.E. (1982). Using Knowledge to Control Tree Searching. *Artificial Intelligence*, Vol. 18, No. 1, pp. 1–51. [58]

Winands, M. (2003). Lines Of Action Page. http://www.cs.unimaas.nl/m.winands/loa. [82]

Yoshioka, T., Ishii, S., and Ito, M. (1999). Strategy Acquisition for the Game Othello Based on Reinforcement Learning. *IEICE Transactions on Information and Systems*, Vol. E82-D, No. 12, pp. 1618–1626. [96]

Zermelo, E. (1913). Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. *Proceedings of the Fifth Congress of Mathematicians (Cambridge 1912)*, pp. 501–504, Cambridge University Press. [32]

# Symbols and Notations

| | |
|---|---|
| $\alpha$, $\bar{\alpha}$ (lower bound(s)) | 38 |
| $\beta$, $\bar{\beta}$ (upper bound(s)) | 38 |
| $B$ (bound) | 47 |
| $\#A$ (cardinality of set A) | 123 |
| $C$ (complexity) | 50 |
| $d$ (depth) | 50 |
| $\epsilon$ (error) | 32 |
| $G$ (game) | 4 |
| $G(h)$ (subgame) | 5 |
| $h$ (history / position / node) | 4 |
| $h_i$, $h + m_i$ (child node) | 50 |
| $\|h\|$ (history length) | 4 |
| $h(k)$ (subhistory) | 4 |
| $H$ (set of histories) | 4 |
| $T$ (learn set of histories) | 123 |
| $H_0$ (MAX's histories) | 26 |
| $H_{op}$ (MIN's histories) | 26 |
| $H_{spec}$ (speculation histories) | 42 |
| $E$ (set of terminal histories) | 4 |
| $E_0$ (MAX's terminal histories) | 26 |
| $E_{op}$ (MIN's terminal histories) | 26 |
| $E_{\omega_i}$ ($\omega_i$'s terminal histories) | 112 |
| L (likelihood) | 123 |
| max $\arg_i$ (operator) | 30 |
| min $\arg_i$ (operator) | 20 |
| $m$, $m_i$ (move) | 5 |
| $m^*$ (best move) | 38 |
| $\bar{m}$ (move vector) | 111 |
| $m(h)$ (move function) | 5 |
| $r(m)$ (move rank) | 17 |
| $M$ (set of moves) | 4 |
| $L$ (move list) | 38 |
| $\Omega(\bar{\pi})$ (opponent model) | 123 |
| $[\omega_0, \ldots, \omega_n]$ (opponent model) | 132 |
| $\Omega$ (target opponent) | 122 |
| $\omega_i$ (opponent type) | 105 |
| $\pi_i$, $\Pr(\omega_i)$ (opp.-type probability) | 105 |
| $\bar{\pi}$ (opp.-type probabilities) | 123 |
| $p$ (player) | 5 |
| MAX (player) | 16 |
| MIN (opponent) | 16 |
| $P_s$ (player's fallibility) | 17 |
| $p(h)$ (player function) | 4 |
| Pr (probability) | 17 |
| $\hat{\Pr}$ (probability estimation) | 136 |
| $\mathbf{s}^*$ (solution) | 5 |
| $G_{min}$ (solution tree for MIN) | 67 |
| $E_{min}$ (solution tree leaves) | 67 |
| $s_p$ (strategy for player) | 5 |
| $s_{mm(V)}$ (minimax strategy) | 27 |
| $s_{om(V_1,V_2)}$ (OM-search strategy) | 27 |
| $b$ (behaviour strategy) | 105 |
| $\sigma$ (mixed strategy) | 104 |
| T (learning time) | 136 |
| $v$, $v_i$ (value) | 38 |
| $v^*$ (best value) | 38 |
| $\bar{v}$, $\bar{w}$ (value vector) | 110 |
| $V$, $V_i$ (value / evaluation function) | 5 |
| $\mathcal{V}$ (set of eval. functions) | 63 |
| $V_0$ (MAX's eval. function) | 25 |
| $V_{op}$ (MIN's (opp.) eval. function) | 25 |
| $v(G)$ (game score) | 5 |
| $v(h)$ (subgame value) | 5 |
| $v_0$ (MAX's value) | 25 |
| $v_{op}$ (MIN's (opp.) value) | 25 |
| $v_{\omega_i}$ ($\omega_i$'s value) | 106 |
| $\mathbb{V}$ (profitability) | 65 |
| $\mathbf{V}$ (true subgame value) | 63 |
| $w$ (width) | 50 |

# Index

# Summary

The thesis deals with the question how opponent models can be used in computer game-playing. The search algorithms normally used in programs that play games like chess are algorithms equivalent with Minimax. They do not use explicit knowledge of the opponent. Yet it seems obvious that knowledge of the opponent can improve game playing. In 1993 two research teams, one in Israel and one in the Netherlands, simultaneously and independently invented a game-tree search method that involved knowledge of the opponent. It is called Opponent-Model search (OM search). This search method is our first main subject. The second main subject is a new search method that also uses an opponent model, but in contrast to the previous model incorporates uncertainty. Therefore, it is called Probabilistic Opponent-Model search (PrOM search).

In the first chapter we provide a short introduction to the research domain, which is the playing of board games by computers, then some relevant notions of mathematical game theory are given together with the basic notions of computer game-playing. The introduction is completed by the following problem statement: *under what conditions can OM search and PrOM search improve computer game-playing?* To answer this statement we formulate four research questions: (1) what are the important theoretical properties of OM search and PrOM search? (2) how can these search methods be implemented efficiently? (3) what is the nature of the practical circumstances in which OM search and PrOM search perform better than the current implementations of Minimax search? and (4) can PrOM search be used in practice to learn an opponent model?

The second chapter is dedicated to related research. We start with a comparison of five foregoing search methods that use some form of opponent model. Then we treat the simultaneous invention of OM search by Carmel and Markovitch in Israel and by Iida, Uiterwijk, Van den Herik, and Herschberg in the Netherlands. The core idea of OM search is that when the strategy of the opponent is known to the player, and the opponent uses Minimax search, then this knowledge can be used to lure the opponent into positions that are profitable for the player. The chapter presents the basic formulation of the search method and discuss the work that has been performed by both research teams. Each of them has developed specialisations and generalisations of OM search. The chapter ends with a discussion of some topics from mathematical game theory that are related to opponent models, such as special equilibria and nonzero-sum games.

The topic of the third chapter is the efficient implementation of OM search. We discuss two basic approaches for implementation: a one-pass approach and an approach that uses $\alpha$-$\beta$ probes. An important factor of efficiency is the number of game positions that can be disregarded when searching for the best move. Therefore efficient *pruning* of the search tree is a main research issue. We discuss several pruning algorithms for OM search for both the one-pass version and the probing version. For one-pass $\beta$-pruning OM search and $\beta$-pruning OM search with $\alpha$-$\beta$ probes, we provide a best-case analysis, showing that the former is the most efficient in the best case. The number of position evaluations that both algorithms need appear to be an almost linear function of the number of evaluations that $\alpha$-$\beta$ search needs. The last part of the chapter discusses the application of a range of search enhancements that are widely used in computer game-playing. Not surprisingly many enhancements are applicable, especially in the versions of OM search with $\alpha$-$\beta$ probes.

In chapter four we discuss the role of evaluation functions in OM search, and especially of errors in these functions. Since it is impossible to investigate all positions of a game, evaluation functions are used in computer game-playing to assess positions at the point where the search is exhausted. The inventors of OM search assume that the own evaluation function should be better than the opponent's function. However, evaluation functions can be interpreted in different ways. The chapter compares three interpretations of evaluation functions and lists eight possible orderings of functions. One of the orderings leads to the definition of estimation errors in evaluation functions. The second part of the chapter shows that these estimation errors can cause serious problems in OM search. We define four types of errors in OM search that follow from estimation errors; two of them are beneficial but the other two are harmful. Especially the type-I error can be dangerous. They appear when the own evaluation function overestimates a position that is assessed correctly by the opponent's evaluation function. In such circumstances OM search tries to lure the opponent into this position, because it seems profitable. The opponent follows willingly, causing the player to be caught in his or her own trap. Therefore, we define a condition, called *admissibility* condition, on the pair of evaluation functions that should prevent such awful circumstances.

After these theoretical chapters, we present in the fifth chapter the results of experiments with OM search in four different game domains. The first domain is random game trees. It is used to measure the average-case behaviour of the two $\beta$-pruning OM-search algorithms mentioned above. The version with $\alpha$-$\beta$ probes appears to be more efficient on the random game trees. We used the same domain to show that transposition tables have a positive effect on the efficiency of $\beta$-pruning OM search with $\alpha$-$\beta$ probes. The second domain is the game lines of action (LOA). The experiments performed show a poor performance of OM search although perfect knowledge of the opponent is available. An explanation is the absence of admissibility of the evaluation functions used. The third game domain is the King-Queen-King-Rook endgame of chess. In the experiments, admissibility is guaranteed since the player has access to an endgame database and the opponent not. The experiments are designed to show whether a player in a lost position could reach a draw or a win with the use of OM search. The results are not conclusive although in some conditions, OM search plays better than $\alpha$-$\beta$ search. The fourth game domain is the mancala game bao. In these experiments, we attempt to discover what exactly is needed to bring OM search to success. It appears that a combination of good opponent prediction and extended search depth is needed for good results. Both factors cause OM search to use more resources. Of these two factors, an extended search depth seems to be more important than a good prediction. For the effect of OM search, the quality of the evaluation functions appears to be important, too.

Chapter six describes PrOM search. The chapter begins with an explanation of the probabilistic opponent model that underlies PrOM search. The model is related to the game-theoretic notion of a mixed strategy. It consists of a series of opponent types, characterized by evaluation functions, and a probability distribution. One of the opponent types is the player self, a mechanism that should prevent the negative effects of overestimations. Next we formulate the PrOM-search method and some of its theoretical properties. We present two different implementations of PrOM search: a one-pass version and a version with $\alpha$-$\beta$ probing. Similar to OM search, pruning is studied in PrOM search, too. It appears that pruning is even more restricted in PrOM search than in OM search. A best-case study reveals that the computational complexity of PrOM search is much higher than that of OM search in the version with $\alpha$-$\beta$ probing and even more in the one-pass version. This is mainly caused by the impossibility of deep pruning in PrOM search. The chapter ends with a discussion on how a probabilistic opponent model can be obtained. Opponent models for PrOM search can in theory be learned from observing the opponent. We concentrate on the learning of the opponent-type probabilities in an off-line and on-line setting.

In the seventh chapter we present experiments with PrOM search, again in four game domains. Experiments in the first domain (random game trees) show that also in the average case, the version of PrOM search with $\alpha$-$\beta$ probing is again more efficient than the one-pass version, but both versions use more much resources than OM search. The experiments show that transposition tables can increase the efficiency of the version with $\alpha$-$\beta$ probes. The second game domain is LOA. The experiments are performed alongside the OM-search experiments above. PrOM search clearly outperforms OM search and $\alpha$-$\beta$ search in these experiments. It indicates that admissibility might be a less strong demand for PrOM search than it is for OM search. The third game domain is a simulated game that we use to test the learning of opponent-type probabilities. The experiments not only confirm some of the expectations from the theoretical analysis, but also give some insight into the dynamics of the learning. It appears that dependencies between opponent types can slow down the learning. The last game domain is bao. We first perform some experiments to learn opponent-type probabilities for a number of probabilistic opponent models. Next we study the performance of PrOM search with different opponent models. The experiments show that PrOM search can perform better in bao than $\alpha$-$\beta$ search even when no perfect knowledge of the opponent is available. However, the gain from using PrOM search is not very large. Furthermore, the search depth was kept constant to six plies for both sides, which means a considerable advantage for PrOM search. The opponent-type probabilities that are learned off-line, appear not to be the best probabilities to use in PrOM search: the probability on the 'self' should be larger than the learned value. Hence, the opponent model should put more weight on the player's own evaluation function than is needed for the prediction of the opponent's moves. A possible explanation for this is that the additional weight balances the negative effects of type-I errors.

The last chapter of the thesis returns to the research questions and the problem statement as formulated in chapter one. From the experiments we learned factors that influence the performance of OM search and PrOM search: the quality of the knowledge and prediction of the opponent, the quality of the evaluation functions, admissibility and the size of the game tree. Taking the answers to the research questions into consideration, the answer to the problem statement must be that OM search or PrOM search can improve computer game-playing, but only in sufficiently profitable circumstances. Both search methods need (far) more resources than the current search methods that are based on $\alpha$-$\beta$ search. Yet, especially OM search suffers from many severe risks. If both methods are applied without serious precautions, the performance of computer game-playing is bound to decrease instead of to improve.

# Samenvatting

Het proefschrift behandelt de vraag hoe opponentmodellen door computerprogramma's kunnen worden toegepast in bordspelen. De zoekmethoden die meestal in programma's voor schaken en andere spelen worden toegepast, gebruiken geen expliciete kennis over de tegenstander (opponent). Het lijkt echter vanzelfsprekend dat kennis over de tegenstander het spel kan verbeteren. In 1993 ontdekten twee onderzoeksgroepen, een in Israël en een in Nederland, tegelijkertijd en onafhankelijk van elkaar een zoekmethode die kennis over de opponent gebruikt. De methode wordt Opponent-Model search (OM search) genoemd. Deze zoekmethode is het eerste hoofdonderwerp. Het tweede hoofdonderwerp is een nieuwe zoekmethode die eveneens een model van de tegenstander gebruikt. In tegenstelling tot OM search wordt in dit model ook onzekerheid over de opponent gemodelleerd. De methode wordt Probabilistic Opponent-Model search (PrOM search) genoemd.

In het eerste hoofdstuk wordt een korte inleiding van het onderzoeksgebied gegeven, namelijk het spelen van bordspelen door computerprogramma's. Dan volgen enige relevante begrippen uit de wiskundige speltheorie en de grondbeginselen van het computerspelen. Na deze introductie komen we tot de volgende probleemstelling: *onder welke condities kunnen OM search en PrOM search het spelen door computers verbeteren?* Teneinde deze probleemstelling te beantwoorden zijn vier onderzoeksvragen opgesteld: (1) wat zijn de belangrijkste theoretische eigenschappen van OM search en PrOM search? (2) hoe kunnen deze methoden efficiënt worden geïmplementeerd? (3) wat is de aard van de omstandigheden waaronder OM search en PrOM search beter presteren dan de huidige implementaties van Minimax search? en (4) kan PrOM search in de praktijk worden gebruikt om een opponent model te leren?

Het tweede hoofdstuk is gewijd aan verwant onderzoek. We beginnen met een vergelijking van vijf voorafgaande zoekmethoden die een of andere vorm van een opponentmodel gebruiken. Daarna behandelen we de gelijktijdige uitvinding van OM search door Carmel en Markovitch in Israël en door Iida, Uiterwijk, Van den Herik en Herschberg in Nederland. De kerngedachte van OM search is dat wanneer de strategie van de tegenstander bij de speler bekend is en de tegenstander gebruik maakt van Minimax, de speler deze kennis kan gebruiken om de tegenstander in posities te lokken die gunstig voor de speler zijn. In het hoofdstuk presenteren we de basisformulering van de zoekmethode en bespreken het werk dat beide onderzoeksgroepen verder aan dit onderwerp hebben gewijd. Elk van de twee groepen heeft specialisaties en generalisaties van de zoekmethode ontwikkeld. Het hoofdstuk eindigt met de bespreking van enkele thema's uit de wiskundige speltheorie die te maken hebben met opponentmodellen, zoals bijzondere evenwichten en niet-nulsomspelen.

Het onderwerp van het derde hoofdstuk is een efficiënte implementatie van OM search. We bespreken twee elementaire wijzen van implementatie: een eengangs-aanpak (one-pass) en een aanpak met $\alpha$-$\beta$-peilingen ($\alpha$-$\beta$-probing). Een bepalende factor voor de efficiëntie is het aantal spelposities dat kan worden overgeslagen tijdens het zoeken naar de beste zet: het efficiënt snoeien (pruning) van de zoekboom is een belangrijk onderzoeksonderwerp. We bespreken verschillende snoeimethoden voor OM search voor zowel de eengangsversie als de versie met peilingen. Voor de belangrijkste twee algoritmen (one-pass $\beta$-pruning OM search en $\beta$-pruning OM search met $\alpha$-$\beta$-probing) geven we een best-case analyse. De analyse toont aan dat het eerste algoritme het efficiënst is in de beste-case. Het aantal positie-evaluaties dat beide algoritmen nodig hebben blijkt een nagenoeg lineaire functie te zijn van het aantal evaluaties dat $\alpha$-$\beta$ search nodig heeft. Het laatste gedeelte van het hoofdstuk bespreekt de toepasbaarheid van een reeks zoekverbeteringen die algemeen gebruikelijk zijn in computerspelen. Menig verbetering is toepasbaar, in het bijzonder in OM search met $\alpha$-$\beta$-probing.

In hoofdstuk vier bespreken we de rol van evaluatiefuncties in OM search en met name de fouten in die evaluatiefuncties. Aangezien het onmogelijk is om alle posities van een spel te onderzoeken gebruiken computerprogramma's evaluatiefuncties om posities te beoordelen op het moment dat het zoeken is uitgeput. De uitvinders van OM search namen aan dat de eigen evaluatiefunctie beter moest zijn dan die van de opponent. Er zijn echter verschillende interpretaties van evaluatiefuncties mogelijk die leiden tot verschillende definities van 'beter'. Het hoofdstuk behandelt drie interpretaties en acht ordeningen van evaluatiefuncties die daaruit voortkomen. Een van de ordeningen leidt tot de definitie van inschattingsfouten in evaluatiefuncties. Het tweede deel van dit hoofdstuk toont aan dat deze inschattingsfouten grote problemen kunnen veroorzaken in OM search. We definiëren vier foutsituaties in OM search; twee van deze zijn gunstig, de andere zijn ongunstig. Met name de type-I fouten zijn gevaarlijk. Zij treden op wanneer de eigen evaluatiefunctie een positie overschat die door de tegenstander correct wordt ingeschat. OM search zal proberen om de tegenstander in deze positie te lokken en de tegenstander zal zonder weerstand volgen - hetgeen ertoe leidt dat de speler in zijn of haar eigen valkuil stapt. We formuleren een voorwaarde die deze situatie moet vermijden: *admissibility*.

Na de theoretische hoofdstukken geven we in hoofdstuk vijf de resultaten van experimenten met OM search in vier domeinen. Het eerste domein is random spelbomen. Het wordt ingezet om het average-case gedrag van de twee bovengenoemde $\beta$-pruning varianten van OM search te meten. Het blijkt dat de versie met $\alpha$-$\beta$-probing efficiënter is met deze random spelbomen. We benutten dit domein tevens om aan te tonen dat transpositietabellen een positief effect hebben op $\beta$-pruning OM search met $\alpha$-$\beta$-probing. Het tweede domein is het spel lines of action (LOA). De experimenten in dit domein tonen een slechte prestatie van OM search ondanks de beschikbaarheid van perfecte kennis over de tegenstander. Een mogelijke verklaring is het ontbreken van admissibility in de betrokken evaluatiefuncties. Het volgende domein is het Koning-Dame-Koning-Toren eindspel in schaken. In deze experimenten kan admissibility gegarandeerd worden omdat de speler een eindspel-database ter beschikking heeft in tegenstelling tot de tegenstander. De experimenten zijn ontworpen om aan te tonen of een speler in een verloren positie remise of winst kan bereiken met OM search. De resultaten zijn niet eenduidig alhoewel in sommige omstandigheden OM search beter speelt dan $\alpha$-$\beta$ search. Het vierde domein is het mancalaspel bao. In deze experimenten proberen we te ontdekken wat precies nodig is om OM search succesvol te maken. Het blijkt dat een combinatie van een goede voorspelling van de tegenstander en grotere zoekdiepte noodzakelijk is voor goede resultaten. Beide factoren veroorzaken dat OM search meer middelen (tijd) nodig heeft. Van deze twee factoren blijkt extra zoekdiepte belangrijker dan een goede voorspelling. Daarnaast is ook de kwaliteit van de gebruikte evaluatiefuncties belangrijk.

Hoofdstuk zes beschrijft PrOM search. Het hoofdstuk begint met een exposé over het probabilistich opponentmodel dat aan PrOM search ten grondslag ligt. Het model is verwant aan het speltheoretisch begrip van gemengde strategie. Het model bestaat uit een reeks opponenttypes die ieder door een evaluatiefunctie worden gekenmerkt, en een kansverdeling. Een van de opponenttypes is de speler zelf, een voorziening die de negatieve effecten van overschattingen moet opheffen. Vervolgens formuleren we de PrOM search zoekmethode en een aantal theoretische eigenschappen. We presenteren twee verschillende implementaties van PrOM search: een one-pass aanpak en een aanpak met $\alpha$-$\beta$-probing. Net zoals bij OM search bestuderen we ook pruning in PrOM search. Het blijkt dat pruning in PrOM search nog beperkter is dan in OM search. Een best-case analyse toont aan dat de computationele kosten van PrOM search veel hoger zijn dan die van OM search, met name in de one-pass versie. Dit wordt voornamelijk veroorzaakt door de onmogelijkheid van zogenaamd diep snoeien in PrOM search. Het hoofdstuk eindigt met de bespreking hoe een probabilistisch opponentmodel verkregen kan worden. We bestuderen met name het leren van kansverdelingen in een off-line en een on-line opstelling.

In het zevende hoofdstuk presenteren we experimenten met PrOM search, opnieuw in vier domeinen. De experimenten in het eerste domein, random spelbomen, laten zien dat ook in average case de versie van PrOM search met $\alpha$-$\beta$-probing efficiënter is dan de one-pass versie, maar beide versies benutten veel meer middelen dan OM search. De experimenten tonen ook aan dat transpositietabellen de efficiëntie van de versie met $\alpha$-$\beta$-probing kan vergroten. Het tweede domein is LOA. Deze experimenten zijn tegelijk met de experimenten hierboven voor OM search uitgevoerd. PrOM search speelt duidelijk beter dan OM search en $\alpha$-$\beta$ search. Dit duidt erop dat admissibility wellicht een minder sterke voorwaarde voor PrOM search is dan voor OM search. Het derde domein is een gesimuleerd spel dat wordt benut om het leren van kansen op opponenttypes te bestuderen. Niet alleen bevestigen de experimenten enkele van de verwachtingen van de theoretische analyse, maar ze geven ook inzicht in de dynamiek van het leren. Het blijkt dat afhankelijkheden tussen opponenttypes het leren vertraagt. Het laatste domein is bao. In dit domein voeren we eerst een reeks experimenten uit om kansen op opponenttypes te bepalen voor een aantal probabilistische opponentmodellen. Daarna bestuderen we de prestaties van PrOM search in een aantal toernooien. De experimenten tonen aan dat PrOM search soms beter presteert dan $\alpha$-$\beta$ search, zelfs wanneer geen perfecte kennis over de opponent aanwezig is. De winst van PrOM search is echter niet groot. Bovendien wordt de zoekdiepte constant gehouden, wat een enorm voordeel voor PrOM search betekent. De kansverdeling over opponenttypes die we hadden geleerd blijkt niet de optimale kansverdelingen te zijn in de toernooien: de kans op het eigen opponent-type blijkt steeds hoger te zijn bij de optimale prestaties. Dit suggereert dat het opponentmodel meer kans moet toekennen aan het eigen opponenttype dan op basis van de zetten van de opponent wordt voorspeld. Een mogelijke verklaring hiervoor is dat de extra kans een tegenwicht is voor de negatieve effecten van type-I fouten.

Het laatste hoofdstuk van dit proefschrift keert terug naar de onderzoeksvragen en de probleemstelling zoals opgesteld in het eerste hoofdstuk. Uit de experimenten leren we de factoren die van invloed zijn op de prestaties van OM search en PrOM search: de kwaliteit van de kennis over de tegenstander en de voorspellingen, de kwaliteit van de evaluatiefuncties, aanwezigheid van admissibility en tenslotte de omvang van de zoekboom. Wanneer we de antwoorden op de onderzoeksvragen in overweging nemen, moet het antwoord op onze probleemstelling luiden dat zowel OM search als PrOM search het spelen door computers kan verbeteren, maar slechts onder zeer gunstige omstandigheden. Beide zoekmethoden benutten veel meer middelen dan het gebruikelijke $\alpha$-$\beta$ search, desalniettemin lijdt met name OM search onder ernstige risico's. Wanneer een van beide methoden zonder ernstige voorzorgsmaatregelen wordt toegepast is het spelen door computers eerder gedoemd te verslechteren dan te verbeteren.

# Curriculum Vitae

Jeroen Donkers was born on April 8, 1963 in Gemert, in the province of Noord-Brabant which is in the southern part of the Netherlands. He started studying biology and mathematics in 1981 at the Katholieke Universiteit Nijmegen and received his baccalaureate (kandidaats) in 1986. After spending almost a year as a master student at the department of cell biochemistry working on mathematical models for membrane proteins, he decided to find a job in computer science. He followed an intensive course in software engineering at Philips in Apeldoorn and started in May 1987 as a scientific programmer and system manager at the department of Medical Informatics of the Universiteit Maastricht.

After eight years of system management, programming, software design, education, and a little scientific work, he moved in May 1995 to the department of Knowledge Systems and the institute MATRIKS at the faculty of General Sciences of the same university. The institute detached him part-time as a scientific programmer to RIKS b.v., but generously allowed him to finish his academic studies in the remaining time at the Knowledge Engineering programme of the faculty. He graduated (doctoraal) in September 1997 cum laude. In January 1998 he received the student award of the faculty for his master thesis.

Meanwhile, he moved within the faculty from MATRIKS to the department of Computer Science and the institute IKAT. After his graduation, he was appointed as a researcher and teacher at the department of Computer Science and worked part-time on his Ph.D. thesis. The first two years he worked on the same subject as his master thesis: decision making under uncertainty. In the autumn of 1999 he decided together with his supervisors to switch to the subject of the current thesis and he joined IKAT's computer-games research group.

During his work at the university he lectured in courses on programming, data structures & algorithms, and databases of the Knowledge Engineering programme. He has been member of the educational board of the faculty and has been educational co-ordinator of the department. As a researcher in IKAT he worked on several commercial projects, especially in the field of international police co-operation. He is responsible for the technical co-ordination of the commercial projects within IKAT.

Although "his work is his hobby", he likes to spent his spare time on many other activities. During the last eight years, however, the emphasis was on playing the piano and on gardening. He lives together with his partner in the city of Maastricht.

# Publications

[1] Donkers, H.H.L.M., Hasman A., Henckens P., and Tange, H. (1989). Strategy, a New Vision on Decision Trees. *Proceedings International Symposium Medical Informatics and Education* (eds. R. Salomon, *et al.*), pp. 224–227. University of Victoria, Canada. ISBN 1-55058-001-9.

[2] Donkers, H.H.L.M. and Hasman, A. (1990). Programming Strategy, Experiences with Object Oriented Turbo Pascal 5.5. *Proceedings Medical Informatics Europe* (eds. O. Rienhoff and D.A.B. Lindberg), Lecture Notes in Medical Informatics, Vol. 40, pp. 411–415, Springer Verlag, Berlin, Germany. ISBN 3-540-52936-5.

[3] Donkers, H.H.L.M.. Westerterp, K., and Hasman, A. (1990). Weight, an Interactive Simulation Model of the Human Energy Housekeeping. *Proceedings Medical Informatics Europe* (eds. O. Rienhoff and D.A.B. Lindberg), Lecture Notes in Medical Informatics, Vol. 40, pp. 430–434, Springer Verlag, Berlin, Germany. ISBN 3-540-52936-5.

[4] Donkers, H.H.L.M. and Hasman, A. (1991). Natural Language Interfacing in Computer Aided Instruction and Testing. *Knowledge Information and Medical Education* (eds. J.H. van Bemmel and J. Zvárová), pp. 269–275, Elsevier Science Publ., Amsterdam, The Netherlands. ISBN 0-444-88902-7.

[5] Donkers, H.H.L.M. and Hasman, A. (1991). Using Object Oriented Turbo Pascal 5.5. *Proceedings Software Engeneering in Medical Informatics*, Amsterdam, pp. 323–338.

[6] Donkers, H.H.L.M., Vleuten, C.P.M. van der, and Schwurith, L.W.T. (1992). Computergestuurd Casusgericht Toetsen. *Proceedings Gezond Onderwijs Congres* (eds. C.P.M. van der Vleten, *et al.*), pp. 138–145. Bohn Stafleu Van Loghum, Houten, The Netherlands. ISBN 9-789031-314898.

[7] Vissers, M.C., Hasman, A., Donkers, H.H.L.M., and Linden C.J. v.d. (1995). Development, Implementation and a First Evaluation of a Protocol Processing System (ProtoVIEW). *Computer methods and programs in biomedicine*. Vol. 47, No. 1, pp. 81–92. ISSN 0169-2607

[8] Schuwirth, L.W.T., Vleuten, C.P.M. van der, Kock, C.A. de, Peperkamp, A.G.W., and Donkers, H.H.L.M. (1996). Computerized Case-Based Testing: A Modern Method to Assess Clinical Decision Making. *Medical teacher : the journal for educators in the health sciences*, Vol. 18, No. 4, pp. 294–299. ISSN 0142-159X.

[9] Schuwirth, L.W.T., Vleuten, C.P.M. van der, and Donkers, H.H.L.M. (1996). A Closer Look at Cueing Effects in Multiple-Choice Questions. *Medical education*, Vol. 30, No. 1, pp. 44-49. ISSN 0308-0110.

[10] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1998). Solving Markov Decision Networks using Incremental Pruning. *Proceedings of the Tenth Netherlands/Belgium Conference on Artificial Intelligence* (eds. H. la Poutré and H.J. van den Herik), pp. 47–156.

[11] Donkers, H.H.L.M., Ferreira, R., Uiterwijk, J.W.H.M., and Herik, H.J. van den (1999). VAS: Quantifying a Qualitative Network. *Proceedings of the Eleventh Belgium-Netherlands Conference on Artificial Intelligence* (BNAIC99) (eds. E.O. Postma and M. Gyssens), pp. 139–146. IKAT, Universiteit Maastricht.

[12] Tange, H.J., Dreesen, V.A.B., Donkers, H.H.L.M., and Hasman A. (1999). A Computer-Based, Communication-Oriented, Medical Record. *International Journal of Healthcare Technology and Management*, Vol. 1 Nos. 3-4, pp. 288–300.

[13] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2000). Investigating Probabilistic Opponent-Model Search. *Proceedings JCIS 2000* (ed. P.P. Wang), pp. 982–985. Association for Intelligent Machinery, Inc. ISBN 0-9643456-9-2.

[14] Irving, G., Donkers, H.H.L.M., and Uiterwijk, J.W.H.M. (2000). Solving Kalah. *ICGA Journal.* Vol. 23, No. 3. pp. 139–148.

[15] Donkers, H.H.L.M., Voogt, A. de, and Uiterwijk, J.W.H.M. (2000). Human versus Machine Problem Solving: Winning Openings in Dakon. *Board Games studies.* Vol. 3, pp. 79–88. ISBN 90-5789-057-7, ISSN 1566-1962.

[16] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2000). Investigating Probabilistic Opponent-Model Search. *Proceedings BNAIC 2000* (eds. A. van den Bosch and H. Weigand), pp. 337–338.

[17] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2001). Probabilistic Opponent-Model Search. *Information Sciences*, Vol. 135, Nos. 3–4, pp. 123–149.

[18] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2001). Admissibility in Opponent Model Search. *Proceedings BNAIC 2001* (eds. B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren), pp. 373-380.

[19] Donkers, H.H.L.M. (2001). Learning Opponent-Type Probabilities for PrOM Search (extended abstract). *The CMG Sixth Computer Olympiad Computer-Games Workshop Proceedings* (ed. J.W.H.M. Uiterwijk). Technical Report Series CS 01-04. IKAT, Department of Computer Science, Universiteit Maastricht. ISSN 0922-8721

[20] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Voogt, A.J. de (2002). Mancala Games. *Step by step. Proceedings of the 4th colloquium 'Board Games in Academia'* (eds. J. Retschitzki and R. Haddad-Zubel) Edition Universitaire, Fribourg, Switzerland. pp. 133-146. ISBN 2-8271-0934-4.

[21] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2002). Admissibility in Opponent-Model Search. *Proceedings JCIS 2002* (eds. P. Wang, *et al.*) pp. 450–453. Association for Intelligent Machinery, Inc. ISBN 0-9707890-1-7.

[22] Donkers, H.H.L.M. and Uiterwijk, J.W.H.M. (2002). Programmming Bao. *Seventh Computer Olympiad: Computer-Games Workshop Proceedings* (ed. J.W.H.M. Uiterwijk), Technical Report Series CS 02-03, IKAT, Department of Computer Science, Universiteit Maastricht.

[23] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2002). Learning Opponent-Type Probabilities for PrOM Search. *BNAIC 2002. Proceedings of the 14th Dutch-Belgian Artificial Intelligence Conference* (eds. H. Blockeel and M. Denecker), pp. 91–98. Leuven, Belgium.

[24] Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den (2003). Admissibility in Opponent-Model Search. *Information Sciences.* Vol. 154, Nos. 3–4, Pp. 119–140.

[25] Donkers, H.H.L.M., Herik, H.J. van den, and Uiterwijk, J.W.H.M., (2003), Opponent-Model Search in Bao: Conditions for a Successful Application, *Advances in Computer Games 10* (eds. H.J. van den Herik, H. Iida, and E. Heinz), Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 307–323.

6 Rogier van Eijk (UU) *Programming Languages for Agent Communication*

7 Niels Peek (UU) *Decision-Theoretic Planning of Clinical Patient Management*

8 Veerle Coupé (EUR) *Sensitivity Analyis of Decision-Theoretic Networks*

9 Florian Waas (CWI) *Principles of Probabilistic Query Optimization*

10 Niels Nes (CWI) *Image Database Management System Design Considerations, Algorithms and Architecture*

11 Jonas Karlsson (CWI) *Scalable Distributed Data Structures for Database Management*

**2001**

1 Silja Renooij (UU) *Qualitative Approaches to Quantifying Probabilistic Networks*

2 Koen Hindriks (UU) *Agent Programming Languages: Programming with Mental Models*

3 Maarten van Someren (UvA) *Learning as Problem Solving*

4 Evgueni Smirnov (UM) *Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets*

5 Jacco van Ossenbruggen (VU) *Processing Structured Hypermedia: A Matter of Style*

6 Martijn van Welie (VU) *Task-Based User Interface Design*

7 Bastiaan Schonhage (VU) *Diva: Architectural Perspectives on Information Visualization*

8 Pascal van Eck (VU) *A Compositional Semantic Structure for Multi-Agent Systems Dynamics*

9 Pieter Jan 't Hoen (RUL) *Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes*

10 Maarten Sierhuis (UvA) *Modeling and Simulating Work Practice BRAHMS: a Multiagent Modeling and Simulation Language for Work Practice Analysis and Design*

11 Tom M. van Engers (VU) *Knowledge Management: The Role of Mental Models in Business Systems Design*

**2002**

1 Nico Lassing (VU) *Architecture-Level Modifiability Analysis*

2 Roelof van Zwol (UT) *Modelling and Searching Web-based Document Collections*

3 Henk Ernst Blok (UT) *Database Optimization Aspects for Information Retrieval*

4 Juan Roberto Castelo Valdueza (UU) *The Discrete Acyclic Digraph Markov Model in Data Mining*

5 Radu Serban (VU) *The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-Concerned Agents*

6 Laurens Mommers (UL) *Applied legal epistemology; Building a Knowledge-based Ontology of the Legal Domain*

7 Peter Boncz (CWI) *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*

8 Jaap Gordijn (VU) *Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas*

9 Willem-Jan van den Heuvel (KUB) *Integrating Modern Business Applications with Objectified Legacy Systems*

10 Brian Sheppard (UM) *Towards Perfect Play of Scrabble*

# SIKS Dissertation Series

**1998**

1 Johan van den Akker (CWI[1]) *DEGAS - An Active, Temporal Database of Autonomous Objects*

2 Floris Wiesman (UM) *Information Retrieval by Graphically Browsing Meta-Information*

3 Ans Steuten (TUD) *A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective*

4 Dennis Breuker (UM) *Memory versus Search in Games*

5 Eduard W. Oskamp (RUL) *Computerondersteuning bij Straftoemeting*

**1999**

1 Mark Sloof (VU) *Physiology of Quality Change Modelling; Automated Modelling of Quality Change of Agricultural Products*

2 Rob Potharst (EUR) *Classification using Decision Trees and Neural Nets*

3 Don Beal (UM) *The Nature of Minimax Search*

4 Jacques Penders (UM) *The practical Art of Moving Physical Objects*

5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

6 Niek J.E. Wijngaards (VU) *Re-Design of Compositional Systems*

7 David Spelt (UT) *Verification Support for Object Database Design*

8 Jacques H.J. Lenting (UM) *Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation*

**2000**

1 Frank Niessink (VU) *Perspectives on Improving Software Maintenance*

2 Koen Holtman (TUE) *Prototyping of CMS Storage Management*

3 Carolien M.T. Metselaar (UvA) *Sociaal-organisatorische Gevolgen van Kennistechnologie; een Procesbenadering en Actorperspectief*

4 Geert de Haan (VU) *ETAG, A Formal Model of Competence Knowledge for User Interface Design*

5 Ruud van der Pol (UM) *Knowledge-Based Query Formulation in Information Retrieval*

[1]Abbreviations: SIKS – Ph.D. School for Information and Knowledge Systems; CWI – Centrum voor Wiskunde en Informatica, Amsterdam; EUR – Erasmus Universiteit, Rotterdam; KUB – Katholieke Universiteit Brabant, Tilburg; TUE – Technische Universiteit Eindhoven; UM – Universiteit Maastricht; UT – Universiteit Twente, Enschede; UU – Universiteit Utrecht; UvA – Universiteit van Amsterdam; VU – Vrije Universiteit, Amsterdam.

11 Wouter C.A. Wijngaards (VU) *Agent Based Modelling of Dynamics: Biological and Organisational Applications*

12 Albrecht Schmidt (UvA) *Processing XML in Database Systems*

13 Hongjing Wu (TUE) *A Reference Architecture for Adaptive Hypermedia Applications*

14 Wieke de Vries (UU) *Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems*

15 Rik Eshuis (UT) *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*

16 Pieter van Langen (VU) *The Anatomy of Design: Foundations, Models and Applications*

17 Stefan Manegold (UvA) *Understanding, Modeling, and Improving Main-Memory Database Performance*

**2003**

1 Heiner Stuckenschmidt (VU) *Ontology-Based Information Sharing in Weakly Structured Environments*

2 Jan Broersen (VU) *Modal Action Logics for Reasoning About Reactive Systems*

3 Martijn Schuemie (TUD) *Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy*

4 Petkovic (UT) *Content-Based Video Retrieval Supported by Database Technology*

5 Jos Lehmann (UvA) *Causation in Artificial Intelligence and Law – A Modelling Approach*

6 Boris van Schooten (UT) *Development and Specification of Virtual Environments*

7 Machiel Jansen (UvA) *Formal Explorations of Knowledge Intensive Tasks*

8 Yong-Ping Ran (UM) *Repair-Based Scheduling*

9 Rens Kortmann (UM) *The Resolution of Visually Guided Behaviour*

10 Andreas Lincke (UT) *Electronic Business Negotiation: Some Experimental Studies on the Interaction between Medium, Innovation Context and Cult*

11 Simon Keizer (UT) *Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks*

12 Roeland Ordelman (UT) *Dutch Speech Recognition in Multimedia Information Retrieval*

13 Jeroen Donkers (UM) *Nosce Hostem – Searching with Opponent Models*

Stellingen

behorende bij het proefschrift
"Nosce Hostem - Searching with Opponent Models"
Jeroen Donkers, 5 december 2003.

1. Alhoewel het motto "Nosce Hostem" (ken de tegenstander) uiteraard van het grootste belang is bij het zoeken met opponent-modellen, blijkt het motto "Nosce Te Ipsum" (ken uzelf) hierbij van even groot belang te zijn aangezien fouten in de eigen evaluatiefunctie ernstige gevolgen kunnen hebben voor de uitkomst van het spel.
   *Hoofdstuk 8 van dit proefschrift.*

2. De zoekmethoden Opponent-Model Search en Probabilistic Opponent-Model Search kunnen slechts onder strenge voorwaarden met succes worden toegepast.
   *Hoofdstuk 8 van dit proefschrift.*

3. Als Huizinga het onderzoek naar spelen had kunnen observeren binnen de kunstmatige intelligentie, zou hij zeker tot andere gedachten zijn gekomen inzake de rol van het spel in de moderne wetenschap.
   *Johan Huizinga (1938) Homo Ludens.*

4. Zowel de mathematisch-economische speltheorie als het onderzoek naar spelen binnen de kunstmatige intelligentie zullen baat hebben bij een nauwe samenwerking.

5. Het onderzoeken van spelen verdient eerder de titel "de Formule I van de kunstmatige intelligentie" dan "de Drosophila melanogaster van de kunstmatige intelligentie".

6. Twee belangrijke taken van het wetenschappelijk onderzoek naar spelen binnen de kunstmatige intelligentie zijn het reconstrueren van reeds uitgestorven spelen en het behoud van spelen die door de opkomst van de mondiale consumptiemaatschappij dreigen uit te sterven.

7. Het is een groot goed dat universiteiten middels derde-geldstroom onderzoek zorg dragen voor kennisoverdracht naar de maatschappij. Het is een schande dat zij voor het uitvoeren van hun kerntaak, het onderwijs, deels afhankelijk zijn geworden van deze derde-geldstroom financiering.

8. Het Internet biedt grote mogelijkheden voor een evenwichtige ontwikkeling van de wereldeconomie. Het verdient daarom te worden beschermd door een instituut als de Verenigde Naties tegen de macht van nationale politiek en multinationale handel.

9. Het vaak onstuimig verloop van politieke verwikkelingen in academische bestuurlijke organen lijkt overeen te stemmen met de aanwijzing dat *Homo sapiens* een hogere genetische verwantschap heeft met *Pan troglodytes* (Chimpansee) dan met *Pan paniscus* (Bonobo).
   *http://www.gate.net/~rwms/primegendist.html.*

10. Kunstmatige intelligentie is als het geluk. Men streeft er voortdurend naar, maar op het moment dat het wordt bereikt wordt het niet meer als zodanig beleefd.

11. Het is terroristen gelukt om het WTC I en II in New York te vernietigen; dat zal niemand lukken met het WTC I en II van J.S. Bach.