Strategies anticipating a difference in search depth using opponent-model search

Xinbo Gao\textsuperscript{a}, Hiroyuki Iida\textsuperscript{b} *, Jos W.H.M. Uiterwijk\textsuperscript{c},
H. Jaap van den Herik\textsuperscript{c}

\textsuperscript{a}School of Electronic Engineering, Xidian University, Xi'an 710071, China
\textsuperscript{b}Department of Computer Science, Shizuoka University, 3-5-1 Jhukoku, Hamamatsu 432-8011, Japan
\textsuperscript{c}Department of Computer Science, Universiteit Maastricht, P.O. Box 616, 6200 MD Maastricht, Netherlands

Abstract

In this contribution we propose a class of strategies which focus on the game as well as on the opponent. Preference is given to the thoughts of the opponent, so that the strategy under investigation might be speculative. We describe a generalization of OM search, called \((D,d)\)-OM search, where \(D\) stands for the depth of search by the player and \(d\) for the opponent’s depth of search. A known difference in search depth can be exploited by purposely choosing a suboptimal variation with the aim to gain a larger advantage than when playing the objectively best move. The difference in search depth may have the result that the opponent does not see the variation in sufficiently deep detail. We then give a pruning alternative for \((D,d)\)-OM search, denoted by \(\alpha^2\) pruning. A best-case analysis shows that \(\alpha^2\) prunes very efficiently, comparable to the efficiency of \(\alpha\beta\) with regard to minimax. The effectiveness of the proposed strategy is confirmed by simulations using a game-tree model including an opponent model and by experiments in the domain of Othello. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Opponent modelling; Speculative play; \(\alpha^2\) pruning; Othello

1. Introduction

The minimax method and all its sophisticated variants have as implicit assumption that the player and the opponent use the same search strategy. Basically this means: (1) the leaves are evaluated by an evaluation function and (2) the values are backed up via a minimax-like procedure. The evaluation function may contain all kind of special features but in essence it evaluates the position (including the use of quiescence search)
according to a set of preset criteria. For instance, a chess program will never change the value of a Knight in the evaluation function, not even when it is informed by outside knowledge that the opponent is quite threatening in the endgame when operating with two Knights. The program’s evaluation function is fixed and not speculative. After a leaf has been evaluated, the minimax back-up procedure is applied. It is as logical and practical as one can think. Other ideas on the backing up of a value are sparse (cf. [11]). In the past, some ideas not suitable for practical application are put forward by Rivest [19]. The only exception implemented in tournament programs lies at the very beginning of the back-up procedure. For instance, if a draw is foreseen as the outcome of the game (e.g., by repetition of positions) and the opponent is considered to be weak, a contempt factor may indicate that playing the second-best move is preferred. This is the most elementary step of opponent modelling. It shows a slight deviation from the two above-mentioned steps of a minimax-like strategy, although one can argue that the deviation is within the evaluation function.

An extension of the idea has been developed in opponent-model search. A grandmaster attempting to understand the intention behind the opponent’s previous moves may employ some form of speculative play, anticipating the opponent’s weak reply [7]. Iida et al. [5, 6] modelled such thinking processes based on possible mistakes by the opponent, and proposed OM search (short for opponent-model search) as a generalized game-tree search model. In OM search perfect knowledge of the opponent’s evaluation function is assumed. This knowledge may lead to the conclusion that the opponent is expected to make a mistake in a given position. As a consequence the mistake may be exploited to the advantage of the player possessing the knowledge. In such an OM-search model, it is implicitly assumed that both players search to the same depth in the game tree.

In actual game playing as seen in Shogi tournaments, we have observed [6] that the two opponents may not only have different evaluation functions, but may also reach different search depths. These observations have led us to propose a generalization of OM search, called \((D, d)\)-OM search, in which the difference of depth is incorporated. The difference in depth is in the name: \(D\) stands for the depth of search of the first player, and \(d\) for the opponent’s.

In Section 2 we characterize \((D, d)\)-OM search by definition. Three assumptions are given explicitly and the \((D, d)\)-OM-search algorithm is described in detail. Then, in Section 3, the characteristics of \((D, d)\)-OM search are elaborated upon, and the relationship between \((D, d)\)-OM search, OM search, and minimax is discussed. Section 4 describes a variant of the \(\alpha\)-\(\beta\) algorithm that prunes branches within \((D, d)\)-OM search, denoted by \(\alpha\)-\(\beta^2\) pruning. Section 5 illustrates the performance of a given speculative strategy with random-tree simulations as well as with experiments in the domain of Othello. How to apply this strategy efficiently to actual game-playing positions is discussed in Section 6. Finally, the main conclusions and some limitations of this speculative strategy are given in Section 7.
2. \((D,d)\)-OM Search

This section provides the relevant definitions and assumptions for \((D,d)\)-OM search. In addition, an example is presented showing how a value at any position in a search tree is computed using \((D,d)\)-OM search. By convention and for clarity of understanding, the two players are distinguished as the max player and the min player. Below, we discuss \((D,d)\)-OM search from the viewpoint of the max player.

2.1. Definitions and assumptions

For \((D,d)\)-OM search we use the following definitions and assumptions.

**Definition 1 (Playing strategy).** A playing strategy is a three-tuple \(\langle D, EV, SS \rangle\), where \(D\) is the player’s search depth, \(EV\) is his static evaluation function and \(SS\) denotes the search strategy, i.e., the way to back up values from the leaves to the root in a search tree.

**Definition 2 (Player model).** A player model is the assumed playing strategy of a player. For any player \(X\) with search depth \(D_X\), static evaluation function \(EV_X\) and search strategy \(SS_X\), we define a player model as \(M_X = \langle D_X, EV_X, SS_X \rangle\).

\((D,d)\)-OM search is discussed under three assumptions. Here OM stands for OM search, and MM for minimax strategy. \(P\) is a given position in which the max player is to move.

**Assumption 3 (The opponent’s strategy).** The min player’s playing strategy is \(M_{\text{min}} = \langle d, EV_{\text{min}}, MM \rangle\), which means that the min player will perform some minimax strategy at any successor of \(P\) and will evaluate the leaf positions at depth \((d + 1)\) in the max player’s game tree using static evaluation function \(EV_{\text{min}}\).

**Assumption 4 (Knowledge about the opponent).** The max player knows the strategy of the min player, \(M_{\text{min}} = \langle d, EV_{\text{min}}, MM \rangle\), i.e., his min player’s model coincides with the min player’s strategy.

**Assumption 5 (Exploiting the knowledge).** The max player employs \(\langle D, EV_{\text{max}}, (D,d)\text{-OM} \rangle\) as playing strategy, which means that he evaluates the leaf positions at depth \(D\) using his static evaluation function \(EV_{\text{max}}\) and backs up the values by \((D,d)\)-OM search.

Like OM search, \((D,d)\)-OM search stems from speculative play as practiced by grandmasters. In a game, a grandmaster acquires and uses the model of the opponent to spot a potential mistake, and then obtains an advantage by anticipating this mistake.
2.2. The algorithm of \((D,d)\)-OM search

In \((D,d)\)-OM search, a pair of values is computed for the positions at and above depth \((d+1)\). One value comes from the opponent model and one from the max player’s model. Below depth \((d+1)\), the max player no longer uses the opponent model. There only one value is computed for each position; it is backed up by minimax search.

Let \(i,j\) from now on range over all immediate successor positions of a node in question. Let a node be termed a max node if the max player is to move, a min node otherwise. According to the assumptions, \(D\) is the search depth of the max player and \(d\) is the search depth of the min player as predicted by the max player. Then the function \(V(P,OM(D,d))\) and \(V(P,MM(d))\) are defined for relevant nodes, where \(V(P,OM(D,d))\) is the value considered by the max player and \(V(P,MM(d))\) is the value for the min player, predicted by the max player:

\[
V(P,OM(D,d)) = \begin{cases} 
\max_i V(P_i,OM(D-1,d-1)) & \text{if } P \text{ is an interior max node}, \\
V(P_j,OM(D-1,d-1)) & \text{with } j \\
\text{such that } V(P_j,MM(d-1)) \\
= \min_i V(P_i,MM(d-1)) & \text{if } P \text{ is an interior min node and } d \geq 0, \\
\min_i V(P_i,OM(D-1,d-1)) & \text{if } P \text{ is an interior min node} \\
EV_{\max}(P) & \text{if } d = 0 \text{ (} P \text{ is a leaf node)}, \\
\end{cases}
\]

\[
V(P,MM(d)) = \begin{cases} 
\max_i V(P_i,MM(d-1)) & \text{if } P \text{ is an interior max node}, \\
\min_i V(P_i,MM(d-1)) & \text{if } P \text{ is an interior min node}, \\
EV_{\min}(P) & \text{if } d = -1 \text{ (} P \text{ is a “leaf” node)}. \\
\end{cases}
\]

The algorithm of \((D,d)\)-OM search is given in pseudocode in Fig. 1.

An example of \((D,d)\)-OM search is shown in Fig. 2. In this search tree two different root values are obtained due to the different models of the players. Using \((3,1)\)-OM search yields a value of 11 and using minimax a value of 9. In this example, the max player may thus achieve a better result by \((3,1)\)-OM search than by minimax; he will select the left branch. For clarity, we reiterate that \(d\) denotes the search depth for the opponent, i.e., the final depth will be reached at depth \(d+1\) in the search tree of the first player. In the example, the nodes at depth 2 thus will be evaluated for both players, while those at depth 3 will only be evaluated for the first player.

Moreover, it is assumed that the player using \((D,d)\)-OM search always searches deeper than the opponent, i.e., that \(D > d\). Cases such as the opponent being modelled by a deep search using a very fast but simplistic evaluation function, and the first
procedure \((D,d)\)-OM\((P,\text{depth})\):

*/ Iterative deepening at root \(P\) */
*/ Two values are returned, according to equations (2) and (1) */
if \(\text{depth} = d + 1\) then begin

/* Evaluate the min-player’s leaf nodes */
\(V_{\text{MM}}[P] \leftarrow \text{Evaluate}(P, \text{min})\)
\(V_{\text{OM}}[P] \leftarrow \text{Minimax}(P, \text{depth})\)
return \((V_{\text{MM}}[P], V_{\text{OM}}[P])\)
end

\(\{P_i | i = 1, \ldots, n\} \leftarrow \text{Generate}(P)\)
/* Expand \(P\) to generate all its successors \(P_i\) */
for each \(P_i\) do begin

/* Back up the evaluated values */
if \(P\) is a max node then begin
/* At a max node both the max player and the min player back up the maximum */
\(V_{\text{MM}}[P] \leftarrow \max_{1 \leq i \leq n} V_{\text{MM}}[P_i]\)
\(V_{\text{OM}}[P] \leftarrow \max_{1 \leq i \leq n} V_{\text{OM}}[P_i]\)
end
else begin /* \(P\) is a min node */
/* At a min node, the min player backs up the minimum and the max player backs up the value of the node selected by the min player */
\(V_{\text{MM}}[P] \leftarrow \min_{1 \leq i \leq n} V_{\text{MM}}[P_i]\)
\(V_{\text{OM}}[P] \leftarrow V_{\text{OM}}[P_{j*}]\)
end
return \((V_{\text{MM}}[P], V_{\text{OM}}[P])\)

end

end

end

Fig. 1. The algorithm of \((D,d)\)-OM search in pseudocode.
Fig. 2. \((D,d)\)-OM search and minimax compared, with \(D = 3\) and \(d = 1\). The numbers beside the circles/boxes represent the back-up values by \((3,1)\)-OM search (upper) and minimax from the min player’s point of view (lower), respectively. The numbers inside the circles/boxes represent the back-up values by minimax from the max player’s point of view. Depths 3 and 2 contain the leaf positions for the max player and the min player, respectively, i.e., these values (in italics) are evaluated statically using the max player’s or the min player’s evaluation function.

player relying on a shallower search but with a very sophisticated evaluation function, are not treated in the above formulation. The incorporation of such cases will not be difficult in practice, but it would make the formal definitions needless complex. Hence we do not consider it in this article.

3. Characteristics of \((D,d)\)-OM search

In this section, some characteristics of \((D,d)\)-OM search are described and compared with those of the minimax strategy. The relation among \((D,d)\)-OM search, OM search and minimax is discussed. It results in two remarks and a theorem relating the root values as produced by \((D,d)\)-OM search and minimax.

3.1. Relations among \((D,d)\)-OM search, OM search and minimax

The algorithm of \((D,d)\)-OM search given in Section 2 shows that the max player performs a minimax search when backing up the static-evaluation-function values from depths \(D\) to \((d + 1)\), while from depths \((d + 1)\) to 1 the max player performs pure OM search. So from the viewpoint of search algorithms, \((D,d)\)-OM search can be considered as a combination of pure OM search and minimax search.

A different view is also possible: all the moves determined by minimax, OM search and \((D,d)\)-OM search take some opponent model into account, i.e., each choice is based on the player’s own model and some opponent model. Accordingly, all the three strategies can be considered as opponent-model-based search strategies. The difference among them lies in the specification of the opponent model.
Table 1
The opponent models used in minimax, OM search and \((D,d)\)-OM search

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>The opponent model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimax</td>
<td>((D - 1, EV_{\text{max}}, MM))</td>
</tr>
<tr>
<td>OM search</td>
<td>((D - 1, EV_{\text{min}}, MM))</td>
</tr>
<tr>
<td>((D,d))-OM search</td>
<td>((d, EV_{\text{max}}, MM))</td>
</tr>
</tbody>
</table>

The opponent models used by the max player in minimax, OM search and \((D,d)\)-OM search are listed in Table 1. We assume that the max player moves first with search depth \(D\) and evaluation function \(EV_{\text{max}}\), i.e., in a game tree the root is a max position.

Table 1 shows that OM search is a generalization of minimax (in which the opponent does not necessarily use the same evaluation function as the max player), and \((D,d)\)-OM search is a generalization of OM search (in which the opponent does not necessarily search to the same depth as the max player). This is more precisely formulated by the following two remarks.

Remark 6. \((D,d)\)-OM search is identical to OM search when \(d = D - 1\).

Remark 7. \((D,d)\)-OM search is identical to minimax when \(d = D - 1\) and \(EV_{\text{min}} = EV_{\text{max}}\).

From the opponent models used in the three search strategies, the one in \((D,d)\)-OM search has the highest flexibility due to the smallest limitation of the opponent’s choice about search depth and evaluation function. So, \((D,d)\)-OM search is the most universal mechanism of the three, and has in principle the largest potential for practical use.

3.2. A theorem on root values

Based on the different back-up procedures of the evaluation-function values, the following characteristic can be derived.

**Theorem 8.** For the root position \(R\) in a game tree we have the following relation:

\[
V(R, OM(D,d)) \geq V(R, MM(D)),
\]

where \(V(R, OM(D,d))\) denotes the value at root \(R\) by \((D,d)\)-OM search and \(V(R, MM(D))\) the one by minimax with search depth \(D\). The theorem is proven by induction on the level in the game tree.

The above theorem implies that if the max player has a perfect opponent model, \((D,d)\)-OM search based on such a model can enable the max player to reach a position that may be better, but will never be worse than the one yielded by the minimax
strategy. In this we follow the commonly-accepted assumption that the deeper the search, the higher the playing strength.

4. $\alpha\beta^2$ Pruning $(D,d)$-OM search

In this section, we introduce an efficient variant of $(D,d)$-OM search, which we call $\alpha\beta^2$ pruning $(D,d)$-OM search.

4.1. $\alpha\beta^2$ Pruning

In games, such as Shogi, chess and Othello, the number of nodes visited by a search algorithm increases exponentially with the search depth. This obviously limits the scope of the search, especially since game-playing programs have to meet external time constraints. Ever since minimax was introduced to game-playing, many techniques have been proposed to speed up the search process. We only mention the general $\alpha\beta$ pruning [13], the null-move procedure for chess [1,2] and ProbCut for Othello [3]. On the basis of $\alpha\beta$ pruning, Iida et al. proposed $\beta$-pruning as an enhancement for OM search [5].

$(D,d)$-OM search backs up the static-evaluation-function values from depths $D$ to $(d + 1)$ with minimax, and from depths $(d + 1)$ to the root with OM search. So it is possible to split $(D,d)$-OM search into two parts, and then speed them up separately. To guarantee generality, we select $\alpha\beta$ pruning to speed up the minimax part and $\beta$-pruning for the OM-search part. The whole algorithm is named $\alpha\beta^2$ pruning.

For details about $\alpha\beta$ and $\beta$ pruning, we refer to [13] and [5] respectively. Pseudocode for the $\alpha\beta^2$ algorithm is given in Fig. 3.

4.2. Analysis of the $\alpha\beta^2$ pruning’s best case

Below we perform a quantitative study of the savings of the $\alpha\beta^2$ pruning algorithm. Otherwise stated, we focus on the question: how many nodes of a tree need to be examined on the average?

We start considering the question on how many game-tree nodes must be examined in the best case. The search costs are assumed to depend mainly on the evaluation (the building of the search tree and the backing-up procedure are assumed to have negligible costs). Therefore, in the discussion below the efficiency is examined by focussing on the counting of statically evaluated positions. Moreover, we assume that the cost of an evaluation, either by a min player or by a max player, has a constant cost of 1 unit. Furthermore, the game tree is assumed to be uniform with $w$ successors at any non-leaf position and to have depth $D$.

Considering the ‘pure’ $(D,d)$-OM search algorithm without any improved efficiency (see Fig. 1), the max player has to evaluate $w^D + w^{d+1}$ positions.

Below, we distinguish three types of max nodes. First, type-1 max nodes are defined recursively: the root node of the search tree is a type-1 node; further, every left-most
successor of every child of a type-1 node is a type-1 node. Second, every brother node of a type-1 node is a type-2 node. Third, all other max nodes in the search tree are type-3 nodes.

We investigate the search tree at level \( d+1 \), called the evaluation level. Here the min-player evaluations are performed. The max-player evaluations may be also performed on this level \( (D = d + 1) \) or are backed-up from larger depths. We distinguish two cases: (1) the evaluation level is a max level \( (d \) is odd) or (2) a min level \( (d \) is even).
4.2.1. $d$ is odd

The number of type-1 nodes at the evaluation level equals $w^{(d+1)/2}$. At every such node one min-player evaluation has to be performed. The number of max-player evaluations depends on the remaining depth $(D - d - 1)$ beneath the evaluation level. Since only max-player evaluations are involved, the remaining trees can be searched up to depth $D$ using $\alpha$-$\beta$. The number of max-player evaluations is thus given by $N_{\alpha\beta}(D - d - 1, w)$, i.e., the costs for an examination by the $\alpha$-$\beta$ algorithm in the best case for the specified depth and width and is given [13] by

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

The total costs for type-1 nodes at the evaluation level thus are given by

$$w^{(d+1)/2}(1 + N_{\alpha\beta}(D - d - 1, w)).$$

As the simplest example, a best-case search tree for (2,1)-OM search and width 2 is given in Fig. 4 with 4 evaluations for the type-1 leaf nodes. An example where the max player looks deeper in the tree than the min player is given in Fig. 5 (a best-case example for (3,1)-OM search and width 2) with a total of 6 evaluations for the type-1 nodes at the evaluation level.
For every type-2 node at the evaluation level, only one min-player evaluation has to be performed. No max-player evaluations are needed, since in best case they are irrelevant for the back-up values of the parent nodes (see Figs. 4 and 5). Since the total number of type-2 nodes at the evaluation level equals \((w - 1)\) times the number of type-1 nodes, the total costs for the type-2 nodes are given by \((w - 1)w^{(d+1)/2}\).

For each type-3 node at the evaluation level we have again one min-player evaluation and no max-player evaluations. Since type-3 nodes have ancestors at which \(\beta\)-pruning has been performed (see Fig. 6), we have to count the number of type-3 nodes at the evaluation level. By discrete summation we find that this number equals

\[
(w - 1) \sum_{k=1}^{(d-1)/2} w^k N_{2\beta}(d - 2k, w),
\]

where \(d \geq 3\). When \(d = 1\) the number of type-3 nodes is \(w(w - 1)\).

Taking together all costs, we obtain, for the case with \(d\) odd,

\[
\begin{cases}
  w^{(d+1)/2}(1 + N_{2\beta}(D - d - 1, w)) + (w - 1)w^{(d+1)/2} \\
  \quad + (w - 1) \sum_{k=1}^{(d-1)/2} w^k N_{2\beta}(d - 2k, w) \\
  = w^{(d+1)/2}(w + N_{2\beta}(D - d - 1, w)) \\
  \quad + (w - 1) \sum_{k=1}^{(d-1)/2} w^k N_{2\beta}(d - 2k, w) \\
  \quad + (w - 1)w^{(d+1)/2} + w(w - 1) \text{ when } d = 1.
\end{cases}
\]
4.2.2. \(d\) is even

When the evaluation level is a min level (\(d\) is even), we distinguish the nodes according to the parent max nodes. The number of type-1 parent nodes now equals \(w^{d/2}\), each with \(w\) min-player evaluations (one for each child) and \(N_{\alpha\beta}(D - d, w)\) max-player evaluations. The total costs for type-1 nodes are thus given by

\[w^{d/2}(w + N_{\alpha\beta}(D - d, w)).\]

Fig. 7 shows that in the simplest case (a best-case example for (3,2)-OM search and width 2) the costs for the type-1 nodes amount to 8. If the max player searches 1 ply deeper, the costs grow to 10 (Fig. 8).

For type-2 and type-3 parent nodes \(\beta\)-pruning has been performed. These nodes have only one child (min node) and its min-player evaluation value can be determined using \(\alpha\beta\) as described in the case for type-3 nodes with \(d\) odd. Hence, the costs for type-2

---

Fig. 7. An example of \(x\beta^2\) (3,2)-OM search. See Fig. 4 for legends.

Fig. 8. An example of \(x\beta^2\) (4,2)-OM search. See Fig. 4 for legends.
Table 2
The best-case costs by four different search algorithms for various search depths and widths. The ratio is an indication of the efficiency of the pruning algorithm.

<table>
<thead>
<tr>
<th>$D$</th>
<th>$d$</th>
<th>$w$</th>
<th>Minimax</th>
<th>$\alpha\beta$</th>
<th>Ratio (%)</th>
<th>$(D,d)$-OM</th>
<th>$\alpha\beta^2$</th>
<th>Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>32</td>
<td>11</td>
<td>34.4</td>
<td>40</td>
<td>16</td>
<td>40.0</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td></td>
<td>48</td>
<td>20</td>
<td>20</td>
<td>41.7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td></td>
<td>64</td>
<td>18</td>
<td>18</td>
<td>28.1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>2</td>
<td>1024</td>
<td>63</td>
<td>6.2</td>
<td>1280</td>
<td>138</td>
<td>10.8</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>2</td>
<td></td>
<td>1536</td>
<td>162</td>
<td>162</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>2</td>
<td></td>
<td>2048</td>
<td>226</td>
<td>226</td>
<td>11.0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>10</td>
<td>100,000</td>
<td>1099</td>
<td>1.1</td>
<td>101,000</td>
<td>1280</td>
<td>1.3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>10</td>
<td></td>
<td>110,000</td>
<td>2900</td>
<td>2900</td>
<td>2.6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>10</td>
<td></td>
<td>200,000</td>
<td>3710</td>
<td>3710</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>10</td>
<td>$10^{10}$</td>
<td>19,999</td>
<td>0.002</td>
<td>$1.01 \times 10^{10}$</td>
<td>577,010</td>
<td>0.006</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>10</td>
<td></td>
<td>1.1 $\times 10^{10}$</td>
<td>730,010</td>
<td>0.007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>10</td>
<td></td>
<td>$2.0 \times 10^{10}$</td>
<td>4,960,010</td>
<td>0.025</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>35</td>
<td>52,500,000</td>
<td>44,099</td>
<td>0.084</td>
<td>52,564,750</td>
<td>46,480</td>
<td>0.088</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>35</td>
<td></td>
<td>54,022,500</td>
<td>127,400</td>
<td>127,400</td>
<td>0.236</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>35</td>
<td></td>
<td>105,043,750</td>
<td>167,860</td>
<td>167,860</td>
<td>0.160</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>35</td>
<td>$2.76 \times 10^{15}$</td>
<td>$1.05 \times 10^{8}$</td>
<td>$3.80 \times 10^{-6}$</td>
<td>$2.76 \times 10^{15}$</td>
<td>312,001,410</td>
<td>$1.1 \times 10^{-5}$</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>35</td>
<td>$2.86 \times 10^{15}$</td>
<td>$460,691,910$</td>
<td>$1.6 \times 10^{-5}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>35</td>
<td>$5.52 \times 10^{15}$</td>
<td>$9,185,325,660$</td>
<td>$1.7 \times 10^{-4}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and type-3 nodes together are given by

$$(w - 1) \sum_{k=1}^{\frac{d}{2} - 1} w^k N_{3p}(d - 2k, w),$$

where $d \geq 4$. When $d = 2$ the costs for type-2 and type-3 nodes are $w(w - 1)$.

Taking together all costs, we obtain, for the case with $d$ even,

$$
\begin{cases}
    w^{d/2}(w + N_{3p}(D - d, w)) + w(w - 1) & \text{when } d = 2 \\
    w^{d/2}(w + N_{3p}(D - d, w)) + (w - 1) \sum_{k=1}^{\frac{d}{2} - 1} w^k N_{3p}(d - 2k, w) & \text{when } d \geq 4.
\end{cases}
$$

Table 2 presents the costs for several best-case search trees, using four different search algorithms. In this table we include data for the cases $w = 2$ (like our example trees), $w = 10$ (typical for games like Othello) and $w = 35$ (typical for chess-like games).

We can see from Table 2 that $\alpha\beta^2 (D,d)$-OM search is a significant improvement over pure $(D,d)$-OM search. For relatively small $d$, it appears that $\alpha\beta^2$ is almost as efficient as $\alpha\beta$. For larger $d$, $\alpha\beta$ outperforms $\alpha\beta^2$.

We note that in the $M^k$ algorithm, the multi-model-based search strategy developed by Carmel and Markovitch [4], a similar pruning mechanism was described as our $\alpha\beta^2$-pruning. However, due to their recursive application of opponent modelling their pruning is not guaranteed to yield always the same result as the non-pruning analogue. Only when the evaluation functions for both players obey certain conditions, in
particular when they do not differ too much, the correctness of their $z\beta^*$ algorithm is proven.

5. Experimental results of $(D,d)$-OM search

In this section, we describe two experiments on the performance of $(D,d)$-OM search, one with a game-tree model including an opponent model and the other in the domain of Othello. The main purpose of these experiments is to confirm the effectiveness of the proposed speculative strategy when a player has perfect knowledge of the opponent model.

5.1. Experiments with random trees

In order to investigate the performance of a search algorithm, a number of game-tree models have commonly been used [14, 15]. However, for OM-like algorithms we need a model including an opponent model. Iida et al. have proposed a game-tree model to measure the performance of OM search and tutoring-search algorithms [8]. On the basis of this model, we built another game-tree model including the opponent model to estimate the performance of $(D,d)$-OM search. As a measure of performance, we use the $H$ value of an algorithm like we did for OM search. With this game-tree model and the $H$ values, the performance of $(D,d)$-OM search is studied.

5.1.1. Game-tree model

The game-tree model we propose for this experiment is a uniform tree. A random score is assigned for each node in the game tree and the scores at leaf nodes are computed as the sum of numbers on the path from the root to the leaf node. This incremental model was also proposed by Newborn [16] and goes back to a scheme proposed by Knuth and Moore [13]. The max player’s score for a leaf position at depth $D$ (say $P^D$) is calculated as follows:

$$EV_{\text{max}}(P^D) = \sum_{k=0}^{D} r(P^k),$$  \hspace{1cm} (4)$$

the min player’s score for a leaf position at depth $(d + 1)$ (say $P^{d+1}$) is calculated as follows:

$$EV_{\text{min}}(P^{d+1}) = \sum_{k=0}^{d+1} r(P^k),$$  \hspace{1cm} (5)$$

where $-R \leq r(\cdot) \leq R$, and $r(\cdot)$ has a uniform random distribution and $R$ is an adjustable parameter. The resulting random numbers at leaf nodes have a normal distribution. Note that the min player uses the same random score $r(\cdot)$ as the max player. It is implied that $EV_{\text{max}} = EV_{\text{min}}$ when $D = d + 1$. In this case, $(D,d)$-OM search is identical to the minimax strategy according to Remark 2.
This game-tree model comes closer to approximating the parent/child behaviour in real game trees and reflects a game tree including models for both players, in which different opponent models are simulated by various search depths \( d \). For this game-tree model, we recognize that the strength of the min player is equal to that of the max player when \( d = D - 1 \) and that the min player has less information from the search tree about a given position when \( d < D - 1 \). Note that we only investigate situations with \( d \leq D - 1 \), since otherwise \((D, d)\)-OM search is unreliable and should not be used.

5.1.2. \( H \) value

In order to estimate the performance of \((D, d)\)-OM search, like OM search, we define the so-called \( H \) value (Heuristic performance value) for the root \( R \) by

\[
H(R) = \frac{V(R, OM(D, d)) - V_{\text{min}}(R, D)}{V_{\text{max}}(R, D) - V_{\text{min}}(R, D)} \times 100.
\]

(6)

Here, \( V(R, OM(D, d)) \) represents the value at \( R \) by \((D, d)\)-OM search and \( V_{\text{min}}(R, D) \) is given by

\[
V_{\text{min}}(P, D) = \min_i \text{EV}_{\text{max}}(P_i), \quad P_i \in \text{all the leaf nodes at depth } D.
\]

(7)

\( V_{\text{max}}(P, D) \) is similarly given by

\[
V_{\text{max}}(P, D) = \max_i \text{EV}_{\text{max}}(P_i), \quad P_i \in \text{all the leaf nodes at depth } D.
\]

(8)

The strategy indicated by (7) obtains the minimum value of the root \( R \) by looking ahead \( D \) plies and the strategy indicated by (8) analogously the maximum value. \( H(R) \) then represents the normalized performance of \((D, d)\)-OM search and can be thought of as a characteristic of the strategy. Although the value of this performance measure remains to be proven, we have confidence in it, since we feel that the scaling applied by using the minimum and maximum values of the leaves sets the resulting performance in appropriate perspective.

5.1.3. Preliminary results on the performance of \((D, d)\)-OM search

To get a feeling for the performance of \((D, d)\)-OM search, several preliminary experiments were performed using the game-tree model proposed above.

As a first experiment, we observe the performance of \((D, d)\)-OM search for various values of \( d \). In this experiment, \( D \) is fixed at 6 and 7, and \( d \) ranges from 0 to \( D - 1 \). A comparison of \((6, d)\)-OM search and minimax is presented in Fig. 9, while \((7, d)\)-OM search and minimax are compared in Fig. 10, all strategies with a fixed branching factor of 5. All curves shown in Figs. 9 and 10 are average results over 100 experiments.

Figs. 9 and 10 show that

\( \bullet \) the results support Theorem 8 and Remark 2. In particular,

\( \circ \) \( d = 0 \) means that the opponent does not perform any search at all. The max player therefore has to rely on minimax.

\( \circ \) when \( d = 5 \) in Fig. 9 and \( d = 6 \) in Fig. 10, i.e., \( d = D - 1 \), the min player looks ahead to the same depth in the search tree as the max player. In this case, the
max player actually performs pure OM search. Since $EV_{max}(P) = EV_{min}(P)$ in our experiments, the conditions laid down in Remark 7 are fulfilled, and $(D,d)$-OM search is identical to minimax.

- the fluctuation in $H$ values of $(D,d)$-OM search for depths $d$ from 1 to $D−1$ hardly seems dependent on the value of $d$. This is explained by the fact that the ratio of mistakes of OM search does not depend on the depth of search, but only on the branching factor [7]. The results may suggest that the fluctuation in $H$ values of $(D,d)$-OM search has a maximum at $d = \lfloor D/2 \rfloor$.

In a second experiment, we have investigated the performance of $(D,d)$-OM search for various values of $D$. In this experiment, $d$ is fixed at 2 and $D$ ranges from 3 to 7. The results are shown in Fig. 11, which is an average result over 100 experiments, again using a branching factor of 5.

Fig. 11 tells us that the $H$ value of $(D,d)$-OM search is greater than that of $D$-minimax. Of course, the gain of $(D,d)$-OM search over $D$-minimax is very small, since $d$ is fixed at 2, which means that OM search is only performed in the upper 2 plies, whereas in the remainder of the search tree minimax is performed. In addition, $(D,d)$-OM search and $D$-minimax show the same fluctuation in $H$ values, a consequence of both using the same evaluation function.
5.2. Othello experiments

In the subsection above, the advantage of \((D,d)\)-OM search over \(D\)-minimax has been verified with random-tree-model simulations. However, simulating tree behaviour is fraught with pitfalls [17]. So, now let us turn to the study of effectiveness of the proposed speculative strategy in real game-playing. Due to the simple rules and relatively small branching factor, Othello is selected as a test bed. We assume that the rules of the game are known. In determining the final score of a game we adopt the convention that empty squares are not counted for any side. The concept net score is used as the difference in number of stones of a completed game, e.g., in a game with a final score 38-25 the first player has a net score of 13.

5.2.1. Experimental design

For an easy comparison, program A with model \(M_A = \langle D, EV, (D, d)\rangle\)-OM) and program B with model \(M_B = \langle D, EV, MM \rangle\) are assumed to play program C with model \(M_C = \langle d, EV, MM \rangle\). The results of A against C compared to those of B against C then serve as a measure of the relative strengths of \((D,d)\)-OM search and \(D\)-MM search. \(EV\) again denotes the evaluation function. To simplify the experiments, we do not consider the influence of the evaluation function for the moment, i.e., we use the same evaluation function for programs A–C.

In the experiments the programs A and B search to the same depth \(D\), whereas program C searches to depth \(d\). The cases \(D = d + 1, d + 2\) and \(d + 3\) are investigated.

5.2.2. Performance measure

Two parameters \(\Delta S\) and \(R_w\) are defined to estimate the performance of \((D,d)\)-OM search and \(D\)-MM search. \(\Delta S\) represents the average net score and \(R_w\) denotes the winning rate of the player. For a given player \(X\), the \(\Delta S(X)\) is given by

\[
\Delta S(X) = \frac{1}{2N} \sum_{j \in \{B, W\}} \sum_{i=1}^{N} S^i_j(X).
\]
Table 3  
The results of programs A and B vs. program C, for $D = d + 1$

<table>
<thead>
<tr>
<th>Programs</th>
<th>Performance measure</th>
<th>$1$</th>
<th>$2$</th>
<th>$3$</th>
<th>$4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A vs. C</td>
<td>Scores</td>
<td>37.4/26.6</td>
<td>35.8/28.2</td>
<td>38.8/25.0</td>
<td>39.2/24.8</td>
</tr>
<tr>
<td></td>
<td>$\Delta S^B(A)$</td>
<td>10.8</td>
<td>7.6</td>
<td>13.8</td>
<td>14.4</td>
</tr>
<tr>
<td></td>
<td>$R_w(A)$</td>
<td>66%</td>
<td>65%</td>
<td>69.5%</td>
<td>73.5%</td>
</tr>
<tr>
<td>B vs. C</td>
<td>Scores</td>
<td>37.4/26.6</td>
<td>35.8/28.2</td>
<td>38.8/25.0</td>
<td>39.2/24.8</td>
</tr>
<tr>
<td></td>
<td>$\Delta S^B(B)$</td>
<td>10.8</td>
<td>7.6</td>
<td>13.8</td>
<td>14.4</td>
</tr>
<tr>
<td></td>
<td>$R_w(B)$</td>
<td>66%</td>
<td>65%</td>
<td>69.5%</td>
<td>73.5%</td>
</tr>
</tbody>
</table>

In this formula, $\Delta S^B(X)$ denotes the net score obtained by player X when he plays with Black. Similarly, $\Delta S^W(X)$ is the analogous number for playing White, and $2N$ represents the total number of games, equally divided over games starting with Black and with White. Therefore, this performance measure offsets the influence caused by having the initiative, which in general is widely believed to be a decisive advantage in White’s favour.

The winning rate of player $X$, $R_w(X)$ is defined as

$$R_w(X) = \frac{n + m}{2N} \times 100\%,$$  
(10)

where $n$ denotes the number of won games when $X$ plays with White, and $m$ is that when $X$ plays with Black.

In our experiments, we let $N = 50$, i.e., a total of 100 games are played for each case.

5.2.3. Preliminary results

Table 3 shows the results for the case $D = d + 1$, where the average scores by 100 games are given in the format $x/y$, with $x$ the number of stones obtained by the first player and $y$ by the opponent.

From Table 3 we see that programs A and B obtain identical scores against program C, in accordance with Remark 7, i.e., that in the case $D = d + 1$ $(D,d)$-OM search is identical to $D$-MM search. In addition, the results indicate that deepening search can confer some advantage. When $D = d + 1$, the average winning rate is approximately 68.5%.

Table 4 lists the results for the case $D = d + 2$, showing that the performance of $(D,d)$-OM search then always is significantly better than that of $D$-MM search by a small margin.

We speculate that the edge of $(D,d)$-OM search over $D$-MM search will increase with a better evaluation function (the present one mainly just counting disks). This is an area for future research.

Table 5 gives the results for the case $D = d + 3$. Again it is clear that $(D,d)$-OM search is stronger than $D$-MM search. However, when $d = 3$, although the winning rate of $(D,d)$-OM search is greater than that of $D$-MM search, the average net gain
of \((D,d)\)-OM search is surprisingly lower. We believe that this also is a result of the use of a simplified evaluation function. Comparing Tables 3–5 we also notice that the benefit of \((D,d)\)-OM search over \(D\)-MM search grows with larger difference in search depth between the opponents. Obviously, OM search is suited to profit as much as possible from defects in the evaluation function, which is precisely the reason why \((D,d)\)-OM search was proposed. Moreover, although the margins are small we see from Tables 3–5 that \((D,d)\)-OM search always is as good as (when \(D=d+1\)) or better (when \(D>d+1\)) than minimax. We feel that the significance of this observation also depends on the evaluation function in use. This will be subject of future research.

6. Applications of \((D,d)\)-OM search

Since \((D,d)\)-OM search stems from grandmaster’s experience, it is implied that the player using this strategy has a higher playing strength. Even then, a grandmaster employs only in some special cases \((D,d)\)-OM search to get some advantage. These include the case that the opponent is really weak, and the case that the grandmaster reaches a bad position. Regarding the former, \((D,d)\)-OM search can help the player win in fewer moves or by more stones. With respect to the latter, the grandmaster has
to wait for mistakes by his opponent, in which case $(D,d)$-OM search can help him to enhance the position.

6.1. The requirements for applying $(D,d)$-OM search

So far, we assumed that the max player’s static evaluation function $EV_{\text{max}}$ is possibly different from the min player’s one $EV_{\text{min}}$. However, it is very difficult to have reliable knowledge of the opponent’s evaluation function to perform $(D,d)$-OM search. Knowledge of the opponent’s search depth (especially when the opponent is a machine) may be more reliable. We therefore restrict ourselves in this section to potential applications of $(D,d)$-OM search for the case $EV_{\text{max}} = EV_{\text{min}}$.

Under this assumption the requirements for applying the proposed $(D,d)$-OM search can be given by the following lemma.

**Lemma 9.** Let $\delta$ be the search depth difference between the max player and the min player in game playing, i.e., $\delta = D - d$. If $\delta \geq 2$, then $(D,d)$-OM search can be applied.

This means that the condition $\delta \geq 2$ gives the minimum depth difference at which it is beneficial to use $(D,d)$-OM search over minimax in order to anticipate on the opponent’s mistakes resulting from its limited search depth.

The detailed proof for the above lemma can be found in [6]. Furthermore, we can estimate in how many ways $(D,d)$-OM search can be applied. Each way of applying $(D,d)$-OM search is completely defined by the players’ search depths $D$ and $d$, where, for definiteness, $D \geq d + 2$ (from Lemma 9 and Definition 2). By simple discrete summation, we find for the number of ways, considering that the min player may, from instance to instance, choose any model with depth at most equal to $d$ and since the max player may respond by choosing his $D$ to match, that

$$N(D,d) = \sum_{i=1}^{d} (D - i - 1) = D \times d - \frac{1}{2}d(d + 3),$$

where $N(D,d)$ denotes the number of ways of applying $(D,d)$-OM search.

6.2. Possible applications

Since $(D,d)$-OM search is a speculative strategy, the reliability depends on the correctness of the opponent model. We admit that it may seem unlikely that such a strategy will be of much practical use in game-playing. However, there are several situations where such a strategy can be of significant support.

One such possible application is in building a tutoring strategy for game playing [8]. In comparison with the pupil, the tutor can be considered a grandmaster. For tutoring to be successful, the tutor should have a clear representation of his pupil. This statement is paramount when classifying tutoring strategies into the wider context of methods possessing a clear picture of their opponents. Tutoring strategies therefore are a special
case of models possessing an opponent model. The balance in tutoring strategies is
delicate: on the one hand it is essential that the tutor has a good model of his pupil.
On the other hand, the give-away move should not be so obvious as to be noticeable
by the person being tutored. Thereby, with the help of \((D,d)\)-OM search, the game
is manipulated in the direction of an interesting position from which the novice may
find a good or excellent move “by accident”; the novice’s interest in the game may
increase, stimulating his progress on the way towards becoming a strong player.

Another possible application is devising a cooperative strategy for multi-agent games,
such as soccer [12], 4-player variants of chess [18] and so on. In such games, \((D,d)\)-
OM search can be used by the stronger player to construct a cooperative strategy
with his partner(s). Here, compared to the weaker partner(s), the stronger one is a
grandmaster, who can apply \((D,d)\)-OM search in order to model his partner(s) play
[10]. One large advantage of such cooperative strategies is that it is much easier to
obtain a reliable partner model than an opponent model.

7. Conclusions and limitations

In this paper, a speculative strategy for game-playing, called \((D,d)\)-OM search, is
proposed using a model of the opponent, in which difference in search depths is ex-
plicitly taken into account. The algorithm and characteristics of this search strategy are
introduced. A more efficient variant, named \(\alpha - \beta^2\), is also proposed and its efficiency
is analyzed. The effectiveness is confirmed by experimental results from random-tree
 simulations and from the Othello domain.

Although the opponent model used by \((D,d)\)-OM search is more flexible than that
by pure OM search, it is difficult to have a reliable estimate of the search depth
and evaluation function of the opponent. Mostly, the max player will only have a
tentative model of the opponent, and as a consequence this will lead to a risk if
the model is not in accordance with the real opponent’s thinking process. Whereas
preliminary experiments indicated that the applicability of OM search is greater for
weaker opponents [9], more work will be needed to investigate whether this holds also
for \((D,d)\)-OM search.

Another point for future research is the recursive application of \((D,d)\)-OM search,
analogous to Carmel and Markovitch’ [4] \(M^*\) algorithm. Assume we use \((4,1)\)-OM
search. In the present implementation the algorithm uses 2-MM search to determine
the max player’s values at depth 2. A better exploitation of the opponent’s weakness
would be to use \((2,1)\)-OM search. The computational costs for this extension should
carefully be weighed against the benefits.

Acknowledgements

This work was supported in part by the Japanese Ministry of Education Grant-
in-Aid for Scientific Research on Priority Area 732. The authors are grateful to the
anonymous referees whose comments have resulted in numerous improvements of a previous version of this article.

References