Pruning Algorithms for Multi-model Adversary Search

David Carmel and Shaul Markovitch
Computer Science Department,
Technion, Haifa 32000,
Israel
carmel@cs.technion.ac.il shaulm@cs.technion.ac.il

Abstract

The Multi-model search framework generalizes minimax to allow exploitation of recursive opponent models. In this work we consider adding pruning to the multi-model search. We prove a sufficient condition that enables pruning and describe two pruning algorithms, $\alpha\beta^*$ and $\alpha\beta^*_{1p}$. We prove correctness and optimality of the algorithms and provide an experimental study of their pruning power. We show that for opponent models that are not radically different from the player's strategy, the pruning power of these algorithms is significant.

1 Introduction

The minimax algorithm [21] and its $\alpha\beta$ version [12] have served as the fundamental decision procedures for zero-sum games since the early days of computer science. The basic assumption behind minimax is that the player has no knowledge about the opponent's decision procedure. In the absence of such knowledge, minimax assumes that the opponent selects an alternative which is the worst from the player's point of view.

However, it is quite possible that the player does have some knowledge about its opponent. Such knowledge may be a result of accumulated experience in playing against the opponent, or may be supplied by an external source. In chess, for example, simple book-learning methods may reveal some knowledge about the opponent's playing style, such as prefering knights over bishops, avoiding piece exchange etc.

How can such knowledge be exploited? Jansen [10, 11] proposes the *speculative play* paradigm where the player incorporates assumptions about the opponent into its decision procedure. In [2] we describe algorithms that incorporate opponent models into an adversary search. The opponent's decision procedure is assumed to be minimax with a different evaluation function. Iida et al. [7, 8] describe similar algorithms. In [3] we describe the OLSY system that acquires a model of its opponent while playing and utilizes this model during its search. The system was tested in the checkers domain, and was able to outperform a minimax opponent with equivalent playing power.

In a previous work [4] we introduced the *multi-model search*, a generalization of the previous algorithms, that can incorporate recursive opponent models where the opponent model includes (recursively) a model of the player. In this paper we study the possibility of adding pruning to the multi-model search. We prove sufficient conditions over the opponent model that enable pruning and describe two multi-model pruning algorithms using opponent models that satisfy these conditions. We prove correctness and optimality of the algorithms and provide an experimental study of their pruning power.

2 Background: Multi-Model Search[1]

We start by assuming that the player has access to an oracle that specifies the opponent's decision for each state of the game. Later, we will replace the oracle by specific algorithms.

Let S be the set of possible game states. Let $\sigma : S \rightarrow 2^S$ be the successor function. Let $\varphi : S \rightarrow S$ be an opponent model that specifies the opponent's selected move for each state. The M value of a position $s \in S$, a depth limit d, a static evaluation function $f : S \rightarrow \Re$, and an arbitrary opponent model φ, is defined as follows:

$$M(s, d, f, \varphi) = \begin{cases} f(s) & d = 0 \\ \max\limits_{s' \in \sigma(s)} (f(s')) & d = 1 \\ \max\limits_{s' \in \sigma(s)} (M(\varphi(s'), d-2, f, \varphi)) & d > 1 \end{cases}$$

The M value of a position s is obtained by generating the successors of s and by applying φ on each of them to obtain the opponent's response.

[1] Most of the material presented in this section previously appeared in [4].

The M value of each of the selected states is then evaluated (recursively), and the maximum value is returned. The M value of a position s with zero depth is defined to be its static value.

Let $MM(s, d, f)$ be the minimax value of a state s returned by a minimax search to depth d using function f. Minimax can be defined as a special case of M since it applies itself as an opponent model using $-f$ as the opponent's evaluation function and one less depth limit. We denote the minimax value of a state s by $M^0_{(\langle f_0 \rangle, d)}(s)$:

$$M^0_{(\langle f_0 \rangle, d)}(s) = M(s, d, f_0, M^0_{(\langle -f_0 \rangle, d-1)}) = MM(s, d, f)$$

The M^n value of a game state is defined as:

$$M^n_{(\langle f_n, \ldots, f_0 \rangle, d)}(s) \quad = \quad M(s, d, f_n, M^{n-1}_{(\langle f_{n-1}, \ldots, f_0 \rangle, d-1)})$$

Thus, $M^1_{(\langle f_1, f_0 \rangle, d)}(s)$ uses minimax with $d-1$ and f_0 as an opponent model, $M^2_{(\langle f_2, f_1, f_0 \rangle, d)}$ uses $M^1_{(\langle f_1, f_0 \rangle, d-1)}$ as an opponent model etc.

Figure 1 shows an example of a search to depth 3 spanned by M^1. Part *(a)* shows the two calls to minimax to simulate the opponent's search. Part *(b)* of the figure shows the two recursive calls to M^1 applied to the boards selected by the opponent. M^1 first simulates the opponent minimax search from node b using f_0. The opponent chooses the move leads to node e. The player then evaluates node e by calling itself recursively from that node using f_1. This call returns a value of 9 to node e. Since the opponent is expected to choose node e,the player assigns node b the value of 9. Similar reasoning shows that node c gets a value of 3. Therefore, the M^1 value of node a is 9.

Next we present the M^* algorithm that returns the M^n value of a game-state. One of the arguments of the algorithm is a structure called *player* which includes information about both the player's evaluation function and its opponent model.

Definition 1 *A* player *is a pair defined as follows:*

1. *Given an evaluation function f_0, $P^0 = (f_0, \perp)$ is a zero-level* player.
2. *Given an evaluation function f_n and an $(n-1)$-level* player O^{n-1}, $P^n = (f_n, O^{n-1})$ *is an n-level* player. O^{n-1} *is called the* opponent model.

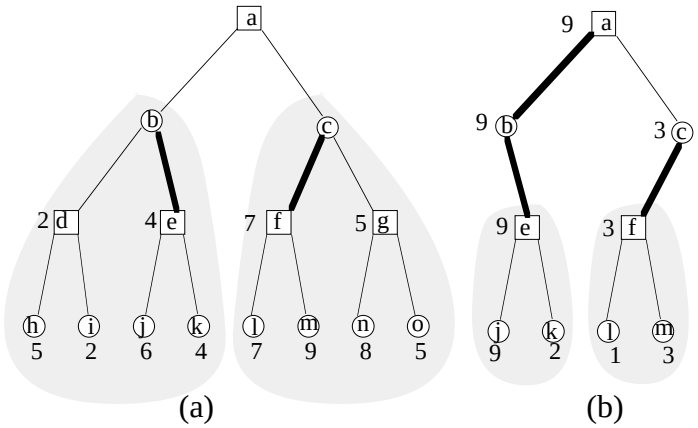

Figure 1: The search tree spanned by M^1 with depth limit 3. Squares represent nodes where it is the player's turn to play. Circles represent nodes where it is the opponent's turn to play. Part (a) shows the two calls to minimax, using f_0, for determining the opponent's choices. Note that the opponent is a maximizer. Part (b) shows the recursive calls of M^1 using f_1, for evaluating the opponent choices.

The recursive definition of a player is in the spirit of the Recursive Modeling Method by Gmytrasiewicz, Durfee and Wehe [6].

The M^* algorithm returns the M^n value of a game-state. It receives a position, a depth limit, and an n-level *player*[2], and determines the M^n value of the game-position and the move selected by the player. The algorithm simulates the opponent's search for each successor in order to anticipate its choice. This simulation is achieved by applying the algorithm recursively with the opponent model as the player. The player then evaluates each of its moves by applying the algorithm recursively on each of the opponent's selections using its own function.

Figure 2 shows an example of a game-tree searched by $M^*(a, 3, f_2(f_1, f_0))$[3]. The recursive calls applied to each node are listed next to the node. The moves selected by the various models are highlighted. The opponent model, (f_1, f_0), simulates the player by using its own model of the player, (f_0), from nodes d and e. At node d the model of the player used by the opponent (f_0) selects node h. At node e, the player's model selects node j (Figure

[2]When the modeling level of the player, n, is smaller than the depth of the search tree, d, we replace the zero-level player $(f_0, \perp)$ by $\overbrace{(f_0, (-f_0, (f_0, \ldots \perp)))}^{d-n}$.

[3]We use $(f_2(f_1, f_0))$ as a shortcut for $(f_2(f_1, (f_0, \perp)))$.

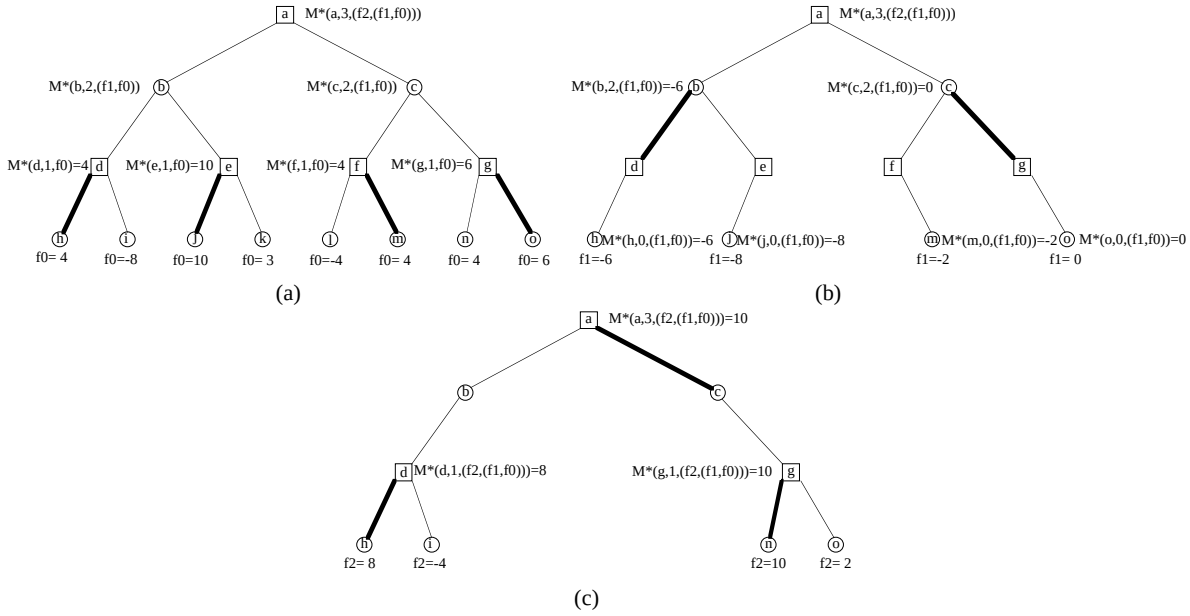

Figure 2: The set of recursive calls generated by calling $M^*(a, 3, (f_2(f_1, f_0)))$. Each call is written next to the node it is called from. (a) The player $(f2, (f_1, f_0))$ calls its model (f_1, f_0) which calls its model of the player (f_0). The moves selected by (f_0) are highlighted. (b) The model (f_1, f_0) evaluates the boards selected by (f_0) using evaluation function f_1. (c) The player evaluates the boards selected by its model using f_2.

2(a)). The opponent then applies its f_1 function to nodes h and j and concludes that nodes h and j, and therefore node d and e, are worth -6 and -8 respectively, (Figure 2(b)). The opponent model, when applied to node b, selects node d. The player then evaluates node d using its own criterion (f_2). It concludes that node d, and therefore node b, are worth 8 (Figure 2(c)). Simulation of the opponent from node c yields the selection of node g. The player evaluates g according to its own strategy and finds that it is worth 10 (the value of node n). The player then selects the move that leads to c with a value of 10. The formal listing of the M^* algorithm is shown in the left part of Figure 3.

It is obvious that the M^* algorithm performs multiple expansions of parts of the search tree. We have developed a *directional* version of the algorithm [17], called M^*_{1p}, that expands the tree one time only. The algorithm expands the search tree in the same manner as minimax does. However, node values

Procedure $M^*(s, d, P^n)$
  if $d = 0$ **then return** $\langle \perp, f_n(s) \rangle$
  else
    $v_n^s \leftarrow -\infty$
    for each $s' \in \sigma(s)$
      if $d = 1$ **then** $v_n^{s'} \leftarrow f_n(s')$
      else
        $\left\langle s'', v_{n-1}^{s'} \right\rangle \leftarrow M^*(s', d-1, O^{n-1})$
        $\left\langle s''', v_n^{s'} \right\rangle \leftarrow M^*(s'', d-2, P^n)$
      if $v_n^{s'} > v_n^s$ **then** $\langle b, v_n^s \rangle \leftarrow \left\langle s', v_n^{s'} \right\rangle$
    return $\langle b, v_n^s \rangle$

Procedure $M_{1p}^*(s, d, P^n)$
  if $d = 0$ **then return** $\langle f_n(s), \ldots, f_0(s) \rangle$
  else
    $v \leftarrow \langle -\infty, \ldots, -\infty \rangle$
    for each $s' \in \sigma(s)$
      $v' \leftarrow M_{1p}^*(s', d-1, P^n)$
      for each active model j
        if $v'_j > v_j$ **then**
          $v_j \leftarrow v'_j$
          if $j < n$ **then** $v_{j+1} \leftarrow v'_{j+1}$
    return $\langle v_n, \ldots, v_d \rangle$

Figure 3: The M^* algorithm (left) and its directional version M_{1p}^* (right).

are propagated differently. Whereas minimax propagates only one value, M_{1p}^* propagates $n+1$ values, $(v_n, \ldots, v_0)$. v_i represents the M^i value of the current node according to the i-level model.

For a given node, we shall denote all the models associated with the player to move as *active models* and all models associated with the player not to move as *passive models*. For values associated with the active models, v_i receives the maximal v_i value among its children. For values associated with the passive models, v_i receives the v_i value of the child that gave the maximal value to v_{i-1}.

Figure 4 shows an example for a tree spanned by M_{1p}^*. Let us look at node e to understand the way in which the algorithm works. Three players

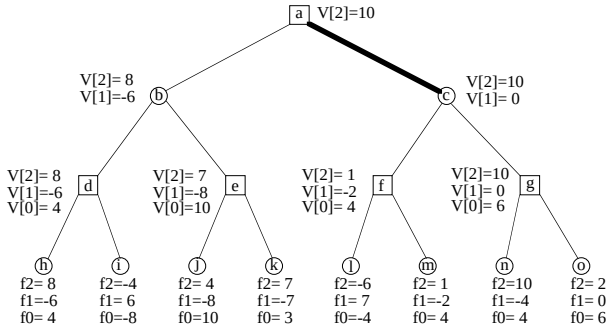

Figure 4: The value vectors propagated by M^*_{1p} for the same game-tree as the one shown in Figure 2.

evaluate node e: the player, the opponent model, and the opponent's model of the player. The opponent's model of the player evaluates all the successors using evaluation function f_0 and selects node j with a value of 10. The opponent model knows that it has no effect on the decision taken at node e, since it is the player's turn to move. It therefore assigns node e the value of node j, selected by the player model, using its own evaluation function f_1, $(f_1(j) = -8)$. The player actually prefers node k with the higher f_2 value $(f_2(k) = 7)$. Thus, the vector propagated from node e is $\langle 7, -8, 10\rangle$. Figure 3 (right) lists the M^*_{1p} algorithm.

The Max^n algorithm for multi-player search [16] also propagates vectors of values up the game-tree but in different manner. In Max^n, at each level of the tree, only one player has an active role in selecting a move, forcing all the other passive players by its own decision. In M^*_{1p}, every model associated with the active player at the current level can make a decision. Therefore, while all the values in Max^n are propagated from the same leaf of the tree, the values in M^*_{1p} can come from different terminal positions. In spite of this difference it is easy to show that for the case of two players with two arbitrary functions, f_1, f_2, the Max^n search can be simulated by M^*_{1p} using the player $(f_1, (f_2, (f_1, (f_2, \ldots \perp))))$.

It seems as though M^*_{1p} is always prefered over M^* because it expands less nodes. M^*_{1p} expands each node in the tree only once, while M^* re-expands many nodes. However, while M^*_{1p} performs less *expansions* than M^*, it may perform more *evaluations*. For example, for the tree shown in Figure 2 and Figure 4, M^* performs 9 expansions and 16 evaluations, while M^*_{1p} performs 7 expansions and 24 evaluations. The following lemma counts

the number of terminal positions evaluated by M^* and M^*_{1p} while searching for the M^n value of a game-tree.

Lemma 1 *The number of evaluations performed by M^* while searching a uniform tree with branching factor b and depth d is*

$$T(b,d) = \frac{\phi_b^{d+1} - \hat{\phi}_b^{\,d+1}}{\sqrt{b^2+4b}}$$

where $\phi_b = \frac{b+\sqrt{b^2+4b}}{2}$, and $\hat{\phi}_b = \frac{b-\sqrt{b^2+4b}}{2}$ [4]*. The asymptotic branching factor of M^* is ϕ_b which converges to $b+1$ for large b.*
The number of evaluations performed by M^*_{1p} is $(d+1)b^d$ [5]*.*

Proof. The number of terminal positions examined by M^* while searching a uniform tree can be established by the following recurrence:

$$T(b,d) = \begin{cases} 1 & \text{if } d = 0 \\ b & \text{if } d = 1 \\ b\,[T(b,d-1) + T(b,d-2)] & \text{otherwise} \end{cases} \tag{1}$$

For $d = 0$ and $d = 1$ the proof is immediate. Assume its correctness for depths less than d.

$$T(b,d) = b\,[T(b,d-1) + T(b,d-2)] = b\left[\frac{\phi_b^d - \hat{\phi}_b^{\,d}}{\sqrt{b^2+4b}} + \frac{\phi_b^{d-1} - \hat{\phi}_b^{\,d-1}}{\sqrt{b^2+4b}}\right] = \tag{2}$$

$$= \tfrac{b}{\sqrt{b^2+4b}}\left[(\phi_b+1)\phi_b^{d-1} - (\hat{\phi}_b+1)\hat{\phi}_b^{\,d-1}\right]$$

ϕ_b and $\hat{\phi}_b$ are both solutions to the equation $x^2 = b(x+1)$. Hence, $b(\phi_b+1) = \phi_b^2$ and $b(\hat{\phi}_b+1) = \hat{\phi}_b^{\,2}$. Substitution of the two equalities into equation 2 completes the proof for the first statement.
For the second statement, it is easy to show by induction on d that $\phi_b^{d-1} \le T(b,d) \le \phi_b^d$. Therefore, $\phi_b^{\frac{d-1}{d}} \le T(b,d)^{\frac{1}{d}} \le \phi_b$.

$$b^* = \lim_{d\to\infty}\left[\frac{\phi_b^{d+1} - \hat{\phi}_b^{\,d+1}}{\sqrt{b^2+4b}}\right]^{\frac{1}{d}} = \phi_b$$

[4] For $b = 1$ we get the Binet formula for Fibonacci sequence.

[5] When the modeling-level n is less than d, the number of evaluations performed by M^*_{1p} is $(n+1)b^d$ (the last $d-n$ evaluations are $f_0(s)$ with alternate signs).

The third statement follows immediately from the inequalities:

$$b + 1 - \frac{1}{b} \leq b^* = \frac{b + \sqrt{b^2 + 4b}}{2} \leq b + 1$$

For M^*_{1p}, any leaf is examined only once, but is evaluated $d+1$ times. □

The above lemma implies that M^* and M^*_{1p} each have an advantage. If it is more important to reduce the number of node expansions than the number of evaluations then one should use M^*_{1p}. Otherwise, M^* should be used. Note that when the set of evaluation functions consists of the same features (perhaps with different weights), the overhead of multiple evaluation is reduced significantly.

3 Pruning in a Multi-model Search

One of the most significant improvements to minimax is the $\alpha\beta$ pruning technique which can reduce the effective branching factor, b, of the tree traversed by the algorithm to approximately $\sqrt{b}$ [12]. In this section we introduce pruning algorithms for the multi-model search.

3.1 Simple pruning

For most practical cases the modeling-level of the player will be smaller than the depth of the search tree. In such cases M^* simulates many zero-level searches at inner nodes of the tree. We have already mentioned that a zero-level search is identical to a minimax search. An obvious improvement for M^* is to simulate these searches by calling $\alpha\beta$. We call such a modification *simple pruning*. A similar pruning method for the one-pass version of one-level search was presented by Iida (β-pruning) [9].

Lemma 2 *Assume that M^* is modified to simulate a zero-level search by an $\alpha\beta$ search. Let P^n be an n-level player. An upper bound on the number of evaluations performed by the modified algorithm while traversing a uniform tree with branching factor b and depth $d > n$, is $(\frac{b}{b-1})^n b^d$. A lower bound on this value is $(\lfloor \frac{d}{2} \rfloor)^n (b^{\lfloor \frac{d+n}{2} \rfloor} + b^{\lceil \frac{d+n}{2} \rceil})$.*

Proof. Let $F_n(b, d)$ be the number of evaluations performed by the modified M^* on a uniform tree with depth $d > n$ and branching factor b. Let $F_0(b, d)$

be the number of positions evaluated by $\alpha\beta$ on such a tree. $F_1(b,d)$ can be established recursively:

$$F_1(b,d) = \begin{cases} 1 & d = 0 \\ b & d = 1 \\ b\left[F_0(b,d-1) + F_1(b,d-2)\right] & \text{otherwise} \end{cases} \quad (3)$$

Upper and lower bounds for $F_0(b,d)$ are b^d and $b^{\lfloor \frac{d}{2} \rfloor} + b^{\lceil \frac{d}{2} \rceil} - 1$ respectively [12]. Repeatedly substituting the upper bound in Equation 3, we get $F_1(b,d) \leq (\frac{b}{b-1})b^d$. An upper bound for $F_n(b,d)$ can be derived inductively from the upper bound for $F_{n-1}(b,d)$: $F_n(b,d) \leq \left(\frac{b}{b-1}\right)^n b^d$.
For the lower bound, repeatedly substituting the lower bound for $F_0(b,d)$ in Equation 3, we get

$$F_1(b,d) \geq \lfloor \frac{d}{2} \rfloor (b^{\lfloor \frac{d+1}{2} \rfloor} + b^{\lceil \frac{d+1}{2} \rceil}) - (\frac{b}{b-1})(b^{\lfloor \frac{d}{2} \rfloor} - 1)$$

A lower bound for a greater modeling level can be approximated by $(\lfloor \frac{d}{2} \rfloor)^n \left(b^{\lfloor \frac{d+n}{2} \rfloor} + b^{\lceil \frac{d+n}{2} \rceil}\right)$.
□

The upper bound for simple pruning, $(\frac{b}{b-1})^n b^d$, is much lower than the number of evaluations performed by M^*, which is approximately $(b+1)^d$. Furthermore, the modified M^* can achieve its lower bound by ordering the successors of the nodes in the zero-level searches according to the best-case order for $\alpha\beta$.

3.2 A Sufficient Condition for Pruning

The simple pruning algorithm prunes only zero-level searches. In this section we discuss the possibility of pruning by higher-level searches.

The $\alpha\beta$ algorithm exploits the strong dependency between the value of a node for the player and its value for the opponent to avoid visiting branches that cannot change the minimax value of the tree. For example, when searching for the minimax value of the tree, shown in the left part of Figure 5, and using f_1, after visiting node f the player concludes that the value of node c for the opponent is greater than -4. Therefore, it is worth at most 4 for the player. Since the value of node b for the player is 8, there is no point in further exploration of node c and node g can be pruned.

M^* cannot infer such constraints on the value of a node for the player based on the node's value for the opponent.'s and t For example, in Figure

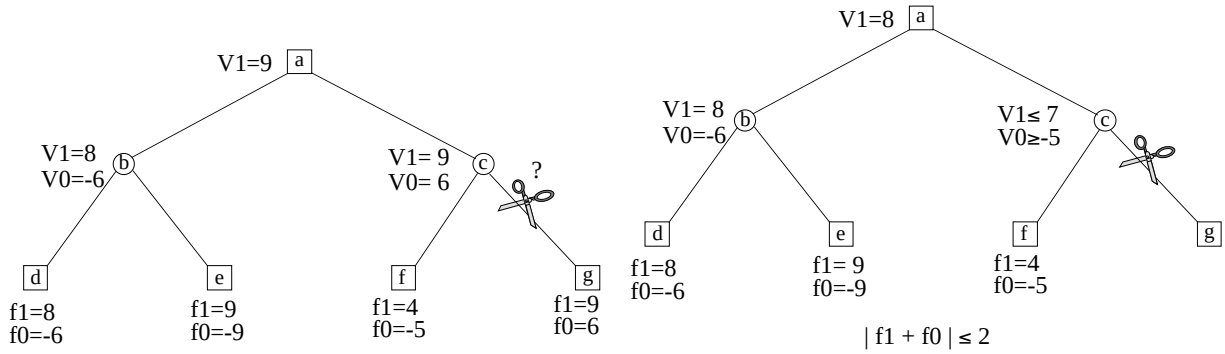

Figure 5: Left: An example of a search tree where $\alpha\beta$, using f_1, would have pruned node g. However, such pruning would change the M^1 value of the tree for the player $P^1 = (f_1, (f_0, \perp))$. $V0$ and $V1$ represent the M^0 and M^1 values of the inner nodes of the tree respectively. Right: Pruning is possible given a bound on the sum of the functions, $|f_1 + f_0| \leq 2$.

5(left), knowing that the opponent will have at least -5 for node c, does not have any implications on the value of node c for the player. Actually, node g determines the $M1$ value of node c and of the entire tree and therefore, pruning is prohibited.

A similar situation arises in *Max*^{n}. Luckhardt and Irani [16] conclude that pruning is impossible in *Max*^{n} without further restrictions about the players' evaluation functions. Korf [15] showed that a shallow pruning for *Max*^{n} is possible if we assume an upper bound on the sum of the players' functions, and a lower bound on every player's function. We use similar restrictions to enable pruning in the multi-model search.

The basic assumption used by the original $\alpha\beta$ algorithm is that $f_1(s) + f_0(s) = 0$ for any game position (the *zero-sum* assumption). This equality holds for any terminal position in zero-sum games and is assumed to be true for non-terminal positions. The modeling framework assumes that non-terminal positions may be evaluated differently by the two players. Figure 5 (left) shows an example where$f_1(g) + f_0(g) = 15$, an unlikely situation since the merit of node g is high for both players. If we assume that the players' function are indeed not too far apart, we can relax the zero-sum assumption to $|f_1(s) + f_0(s)| \leq B$, where $B \geq 0$ is any positive bound (the *bound-sum* assumption). In such a case, although the player's value is not a direct opposite of the opponent's value, we can infer a bound on the player's value based on the opponent's value and B.

Figure 5 (right) shows a search tree similar to the left tree in the figure with one difference: every leaf l satisfies the bound constraint $|f_1(l)+f_0(l)| \leq 2$. In this case, after searching node f, the player can induce that node c is worth at least -5 for the opponent and therefore at most 7 for the player. The player has already a value of 8 for node a. Thus, node g can be pruned.

The *bound-sum* assumption can be used in the context of the M^* algorithm to determine a bound on $f_i + f_{i-1}$ for any game position and therefore for any leaf of the tree. But to enable pruning at any node of the tree, we first need to determine bounds on the sum of $v_i + v_{i-1}$ for inner nodes, i.e., how these sum-bounds are propagated up the search tree.

Lemma 3 *Let s be a node in a game tree and let $s_1, \ldots, s_k$ be its successors. If there exist non-negative bounds $B_1, \ldots, B_n$, such that for each successor s_j of s, and for each model i,*
$|M^i(s_j) + M^{i-1}(s_j)| \leq B_i$, then,
$|M^i(s) + M^{i-1}(s)| \leq B_i + 2B_{i-1}$.

Proof. Assume $M^i(s) = M^i(s_j)$ and $M^{i-1}(s) = M^{i-1}(s_k)$. If $j = k$, $M^i(s)$ and $M^{i-1}(s)$ were propagated from the same successor. Therefore, $|M^i(s) + M^{i-1}(s)| \leq B_i \leq B_i + 2B_{i-1}$. If $j \neq k$, s is a node where i is an active model and $i-1$ is a passive model. Therefore, $M^{i-1}(s)$ and $M^{i-2}(s)$ were propagated from the same successor s_k. Since $i-2$ is an active model at s, $M^{i-2}(s_j) \leq M^{i-2}(s_k)$. Hence, $M^{i-1}(s_k) + M^{i-2}(s_j) \leq M^{i-1}(s_k) + M^{i-2}(s_k) \leq B_{i-1}$. It is easy to show that for any successor s_j, $|M^i(s_j) - M^{i-2}(s_j)| \leq B_i + B_{i-1}$. Summing up the two inequalities for s_j, we get $M^i(s) + M^{i-1}(s) = M^i(s_j) + M^{i-1}(s_k) \leq B_i + 2B_{i-1}$.
For the second side of the inequality, $M^i(s)+M^{i-1}(s) = M^i(s_j)+M^{i-1}(s_k) \geq M^i(s_k) + M^{i-1}(s_k) \geq -B_i \geq -B_i - 2B_{i-1}$. □

According to the lemma, the sum-bounds for each node in the tree at depth d, depend only on the sum-bounds for the nodes at depth $d+1$. Let B_n^d be the bound on $|M^n(s) + M^{n-1}(s)|$, where s is a node at depth d. The bounds for any depth d and any model j can be computed recursively before initiating the search:

$$B_j^d = \begin{cases} B_j & d = 0 \\ B_j^{d-1} & j \text{ is passive at depth } d \\ B_j^{d-1} + 2B_{j-1}^{d-1} & \text{otherwise} \end{cases} \quad (4)$$

Figure 6 illustrates the propagation of the sum-bounds up in the tree. The active models are P^2 and P^0. P^0 (and therefore P^1) propagates its

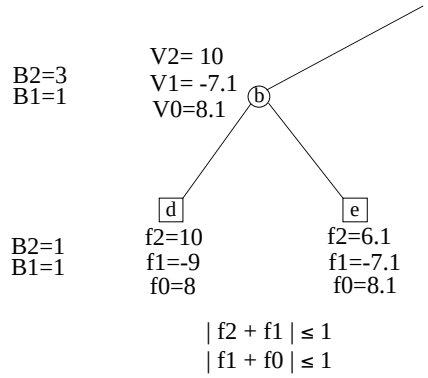

Figure 6: An example for the propagation of the sum-bounds up in the search tree. The active models are P^2 and P^0. P^0 (and therefore P^1) propagates its value from node e. P_2 propagates its value from node d. The theoretical sum-bound computed by the bound lemma is $B_2^1 = B_2^0 + 2B_1^0 = 3$. Indeed, at node b, $V_2 + V_1 = 2.9$.

value from node e. P_2 propagates its value from node d. The theoretical sum-bound computed by Lemma 3 is $B_2^1 = B_2^0 + 2B_1^0 = 3$. Indeed, at node b, $V_2 + V_1 = 2.9$. To violate the sum-bound of 3 at node b, either V_2 should be greater than 10.1 or V_1 smaller than -7.0. But $f_2(d) > 10.1$ implies $f_1(d) < -9.1$ and $f_0(d) > 8.1$. In such a case, P^0 (and therefore P^1) would have selected node d. Similar reasoning applies for $f_0(e) < -7$.

The sum-bounds increase with the distance from the leaves and reduce the amount of pruning. Therefore, using loose bounds will probably prohibit pruning. Note however that for a one-level player, the opponent model is minimax with a sum-bound of zero. Hence, the sum-bound for a one-level player does not increase with depth.

3.3 $\alpha\beta^*$: A pruning version of M^*

Based on lemma 3, we have developed an algorithm, $\alpha\beta^*$, that searches only necessary branches of the tree, assuming given bounds on the sums of the M^n values of the tree nodes. $\alpha\beta^*$ takes as input a position s, a depth limit d, an n-level player P^n, a lower bound α and an upper bound β, and returns the M^n value of s. The formal listing of $\alpha\beta^*$ is shown in figure 7.

3.3.1 The $\alpha\beta^*$ algorithm

The $\alpha\beta^*$ algorithm works similarly to M^*, but, in addition, it computes lower and upper cutoff bounds for the recursive calls and makes pruning decisions:

1. Let s be the current position. The first recursive call computes the opponent selection for the successor s'. The upper bound for this call is $B_n^{d-1} - \alpha$ since if $v_{n-1}(s') \geq B_n^{d-1} - \alpha \geq B_n^{d-1} - v_n(s)$, then $v_n(s) \geq B_n^{d-1} - v_{n-1}(s') \geq v_n(s')$ and s' cannot affect the M^n value of s. For the lower bound, according to Lemma 3, $-B_n^{d-1} - v_n(s') \leq v_{n-1}(s')$. Since $v_n(s') \leq v_n(s) \leq \beta$, the lower bound for the call is $-B_n^{d-1} - \beta$.

2. The value returned by this call, $v_{n-1}(s')$, allows the player to update its lower bound α on the M^n value of s since $v_n(s) \geq v_n(s') \geq -B_n^{d-1} - v_{n-1}(s')$. At this point, if $\alpha \geq \beta$ then the player's value for this branch is higher than its upper bound and there is no reason for searching further.

3. The second recursive call computes the M^n value of the state selected by the opponent, s''. The bounds for this call can be computed similarly to the computation for the first call using the opponent's value $v_{n-1}(s') = v_{n-1}(s'')$. By Lemma 3, $-B_n^{d-2} - v_{n-1}(s') \leq v_n(s'') \leq B_n^{d-2} - v_{n-1}(s')$. In addition, node s'' inherits α and β from s. Since we are interested in the most tight bounds, the bounds for this call are $\alpha_2 = \max(\alpha, -B_n^{d-2} - v_{n-1}(s'))$ and $\beta_2 = \min(\beta, B_n^{d-2} - v_{n-1}(s'))$. Obviously, if $\alpha_2 \geq \beta_2$, we avoid the second recursive call.

4. The rest of the algorithm is identical to $\alpha\beta$. The value returned from the second call is used to update the lower bound α and if $\alpha \geq \beta$, there is no reason to continue searching other successors of s.

3.3.2 Correctness of $\alpha\beta^*$

Theorem 1 *Let P^n be an n-level player. Let $B_n, \ldots, B_1$ be non-negative numbers such that for every game position s and $n \geq j \geq 1$ $|f_j(s) + f_{j-1}(s)| \leq B_j$. Then*

$$\alpha\beta^*(s, d, P^n, -\infty, +\infty) = M^*(s, d, P^n) = M^n_{(\langle f_n, \ldots, f_0 \rangle, d)}(s)$$

```
Procedure αβ*(s, d, P^n, α, β)
  if d = 0 then return ⟨⊥, f_n(s)⟩
  else
    v_n ← α
    for each s' ∈ σ(s)
      ⟨s'', v_{n-1}⟩ ← αβ*(s', d − 1, O^{n-1}, −B_n^{d-1} − β, B_n^{d-1} − α)
      α ← max(α, −B_n^{d-1} − v_{n-1})
      if α ≥ β then return ⟨s', α⟩
      if d = 1 then
        ⟨s''', v'_n⟩ ← ⟨s', f_n(s')⟩
      else
        α_2 ← max(α, −B_n^{d-2} − v_{n-1})
        β_2 ← min(β, B_n^{d-2} − v_{n-1})
        if α_2 < β_2 then
          ⟨s''', v'_n⟩ ← αβ*(s'', d − 2, P^n, α_2, β_2)
        else v'_n ← α_2
      if v'_n > v_n then
        ⟨b, v_n⟩ ← ⟨s', v'_n⟩
      α ← max(α, v_n)
      if α ≥ β then return ⟨b, α⟩
    return ⟨b, v_n⟩
```

Figure 7: The $\alpha\beta^*$ algorithm

Proof. $\alpha\beta^*$ works similarly to M^* but, in addition, it updates cutoff values and makes pruning decisions. For proving that $\alpha\beta^*$ returns the M^n value, it is sufficient to show that any node pruned by $\alpha\beta^*$ can have no effect on the M^n value of its ancestors. There are three types of pruning applied by $\alpha\beta^*$, demonstrated by Figure 8.

1. After computing the decision of the opponent for successor s', $\langle s'', v_{n-1}(s')\rangle$: In this case $\alpha\beta^*$ prunes when the lower bound exceeds the upper bound, $\beta \leq \alpha = -B_n^{d-1} - v_{n-1}(s') \leq -B_n^{d-2} - v_{n-1}(s'') \leq v_n(s'') \leq v_n(s)$. Since $v_n(s) \geq \beta = B_{n+1}^d - \alpha_p$, where α_p is the lower bound on the M^{n+1} value of p, the parent of node s, $\alpha_p \geq B_{n+1}^d - v_n(s) \geq v_{n+1}(s)$. Therefore, node s cannot affect the M^{n+1} value of its parent and can be pruned.

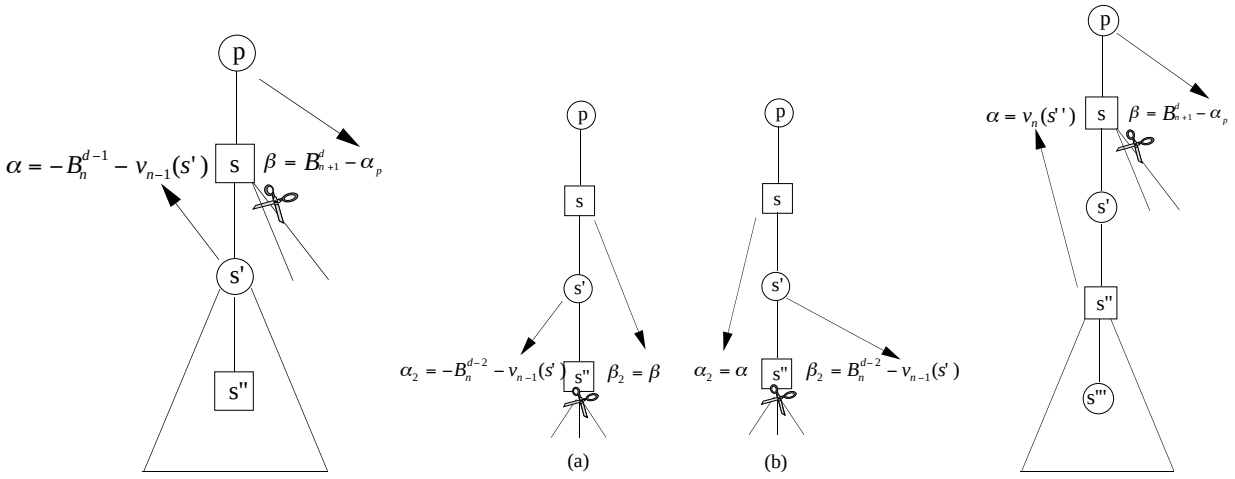

Figure 8: Three types of pruning applied by $\alpha\beta^*$. The arrows indicate the nodes which effect the bounds for pruning. (Left) After computing the decision of the opponent for successor s'. (Middle) Before calling the second recursive calls. (Right) After computing the player's value of the opponent's decision.

2. Before calling the second recursive call, i.e., after computing α_2 and β_2 and $\alpha_2 \geq \beta_2$. There are two possibilities:

   (a) $\alpha_2 = -B_n^{d-2} - v_{n-1}(s')$ and $\beta_2 = \beta$. In this case $v_n(s) \geq v_n(s'') \geq -B_n^{d-2} - v_{n-1}(s'') = -B_n^{d-2} - v_{n-1}(s') = \alpha_2 \geq \beta_2 = \beta$, and as in the former cases, $v_n(s) \geq \beta$ and s can not affect its parent value.

   (b) $\alpha_2 = \alpha$ and $\beta_2 = B_n^{d-2} - v_{n-1}(s')$. In this case $\alpha = \alpha_2 \geq \beta_2 = B_n^{d-2} - v_{n-1}(s') = B_n^{d-2} - v_{n-1}(s'') \geq v_n(s'')$. Since $v_n(s) \geq \alpha \geq v_n(s'')$, the value of s'' for the player is too low and cannot affect $v_n(s)$.

3. After computing the player's value of the opponent's decision, $\langle s''', v_n(s'')\rangle$. In this case $\alpha\beta^*$ prunes if $\alpha = v_n(s'') \geq \beta$. As in the former case, $v_n(s) \geq v_n(s'') \geq \beta$ and node s cannot affect the M^{n+1} value of its parent.

Since only nodes that cannot affect their ancestors are pruned, $\alpha\beta^*$ returns the M^n value of s. □

We have already mentioned that M^* reduces to minimax when the player uses its own function with a negative sign as a model for its opponent.

This is identical to the case of using zero bounds. The following corollary establishes the behavior of $\alpha\beta^*$ when all bounds are zero.

Corollary 1 *For zero sum-bounds $\alpha\beta^*$ reduces to $\alpha\beta$.*

Proof. When all the sum-bounds are zero the n- level player is reduced to $P^n = (f_n, (-f_n, (\ldots \perp))\ldots)$. The recursive call of $\alpha\beta^*$ to simulate the opponent search is $\alpha\beta^*(s', d-1, O^{n-1}, -\beta, -\alpha)$, exactly as $\alpha\beta$ calls itself (in the NEG-MAX version of the algorithm [17]). The second recursive call will not take place since for any successor s', $\alpha_2 \geq 0 - v_{n-1}(s') \geq \beta_2$, and $-v_{n-1}(s')$ will be returned as the player's value for this successor. The rest of the algorithm is identical to $\alpha\beta$. □

The effectiveness of $\alpha\beta^*$ depends on the sum-bounds. For tighter sum-bounds the player's functions become more similar to its model's functions (but with an opposite sign). The amount of pruning increases up to the point where they use the same functions in which case $\alpha\beta^*$ prunes as much as $\alpha\beta$. For loose sum-bounds, the player's functions can be less similar to the opponent's functions and the amount of pruning decreases. For infinite sum-bounds $\alpha\beta^*$ performs only simple pruning since only the zero-level searches can still prune.

Knuth and Moore [12] have shown that for any uniform game-tree there is an order of search that guarantees that $\alpha\beta$ performs $b^{\lfloor \frac{d}{2} \rfloor} + b^{\lceil \frac{d}{2} \rceil} - 1$ evaluations (the lower bound for computing the minimax value of a tree). Is there such a best order for $\alpha\beta^*$? In the general case, the answer is negative. Assume that a player P^d traverses a tree T to depth d. Assume that for each model j, $B_j > |\max_{s \in leaves(T)} f_j(s) + \max_{s \in leaves(T)} f_{j-1}(s)|$. For such trees, $\alpha\beta^*$ must perform all evaluations done by M^* since, at any intermediate stage of the search, there is a possibility for expanding a better leaf and therefore all branches must be searched.

When the modeling level of the player, $P^n = (f_n, (\ldots, f_0))$, is smaller than the depth of search, d, we replace the n-level player by a d-level player, adding a tail of $d-n$ functions with alternating signs, $P^d = (f_n, (\ldots, f_0, -f_0 \ldots, f_0))$. In such a case, all the $d-n$ deeper sum-bounds are zero and hence all these $d-n$ deeper searches are reduced to $\alpha\beta$ according to Corollary 1. Therefore, $\alpha\beta^*$ performs all the pruning done by the simple pruning algorithm when using the same player P^n. In addition, $\alpha\beta^*$ may prune more using the sum-bounds for the higher-level models. Thus, $(\lfloor \frac{d}{2} \rfloor)^n \left(b^{\lfloor \frac{d+n}{2} \rfloor} + b^{\lceil \frac{d+n}{2} \rceil} \right)$,

the best-case for simple pruning, is an upper bound on the best-case performance of $\alpha\beta^*$.

Knuth and Moore have also shown the optimality of $\alpha\beta$ in the sense that any other directional algorithm that searches for the minimax value of a tree in the same order as $\alpha\beta$ does, must evaluate any leaf evaluated by $\alpha\beta$. M^*, and its pruning version $\alpha\beta^*$, are not directional algorithms. However, M^*_{1p} is directional. In the following subsection we describe a pruning version of M^*_{1p} and show its optimality.

3.4 $\alpha\beta^*_{1p}$: A pruning version of M^*_{1p}

In this subsection we describe $\alpha\beta^*_{1p}$, an algorithm that incorporates pruning into M^*_{1p}. $\alpha\beta^*_{1p}$ takes as input a position s, a depth limit d, an n-level player P^n, and for each model i, a lower bound α_i and an upper bound β_i. It returns the M^n value of s.

3.4.1 The $\alpha\beta^*_{1p}$ algorithm

M^*_{1p} works similarly to M^*, but executes the searches of all the models in parallel. $\alpha\beta^*_{1p}$ works like M^*_{1p}, but in addition, it updates cutoff values and makes pruning decisions based on the agreement of all the models. Any active model has the right to veto a pruning decision. The $\alpha\beta^*_{1p}$ algorithm is listed in Figure 9.

1. At first, the algorithm computes new vectors of cutoff values, α', β', from the cutoff values inherited from its parent. If model $j+1$ has decided to prune at the parent node, i.e., $\alpha_{j+1} \geq \beta_{j+1}$, it propagates these cutoff values for model j. Otherwise, the algorithm uses the same mechanism as in $\alpha\beta^*$, $\alpha_j = -B^d_{j+1} - \beta_{j+1}$, and $\beta_j = B^d_{j+1} - \alpha_{j+1}$.

2. The vector of values returned by the recursive call is used for updating the lower cutoff vector α'. Every active model j updates its lower bound if v'_j is greater than its current α'_j.

3. At this point the algorithm makes its pruning decision. Every active model j that has a modeler (i.e. $j < n$) decides to prune by testing whether its value cannot change the modeler's selection in the parent node, i.e., $\alpha_j \geq \beta_j$. If all the active models agree to prune, there is no reason to continue searching other successors of s and the algorithm can return.

```
Procedure αβ*_1p(s, d, P^n, ⟨α_n, ..., α_0⟩, ⟨β_n, ..., β_0⟩)
  if d = 0 then return ⟨f_n(s), ..., f_0(s)⟩
  else
    v ← ⟨−∞, ..., −∞⟩
    α'_n ← α_n
    β'_n ← B^d_n − α_{n−1}
    for each 0 ≤ j < n
      if α_{j+1} < β_{j+1}
        α'_j ← −B^d_{j+1} − β_{j+1}
        β'_j ← B^d_{j+1} − α_{j+1}
      else
        α'_j ← α_{j+1}
        β'_j ← β_{j+1}
    for each s' ∈ σ(s)
      v' ← αβ*_1p(s', d − 1, P^n, ⟨α'_n, ..., α'_0⟩, ⟨β'_n, ..., β'_0⟩)
      for each active model j
        if v'_j > v_j then
          v_j ← v'_j
          if j < n then v_{j+1} ← v'_{j+1}
        α'_j ← max(α'_j, v_j)
      if for every active model n > j ≥ d    α'_j ≥ β'_j
        return ⟨v_n, ..., v_d⟩
    return ⟨v_n, ..., v_d⟩
```

Figure 9: The $\alpha\beta^*_{1p}$ algorithm

In Figure 10 (left), the player already has a value of 8 for node a. Therefore, it sets an upper limit on the value of node c for the opponent: $\beta_0 = B^1_1 - \alpha_1 = 2 - 8 = -6$. The opponent obtains a value of -5 which is greater than its upper bound. Therefore, node g is pruned.

Figure 10 (right) shows pruning performed at a deeper level of the tree. The player P^0 is ready to prune node j as in the left part of the figure. However, to stop the search, the player P^2 must also agree to prune. P^2 indeed decides to prune, using its β_2 which was inherited from its great grandparent.

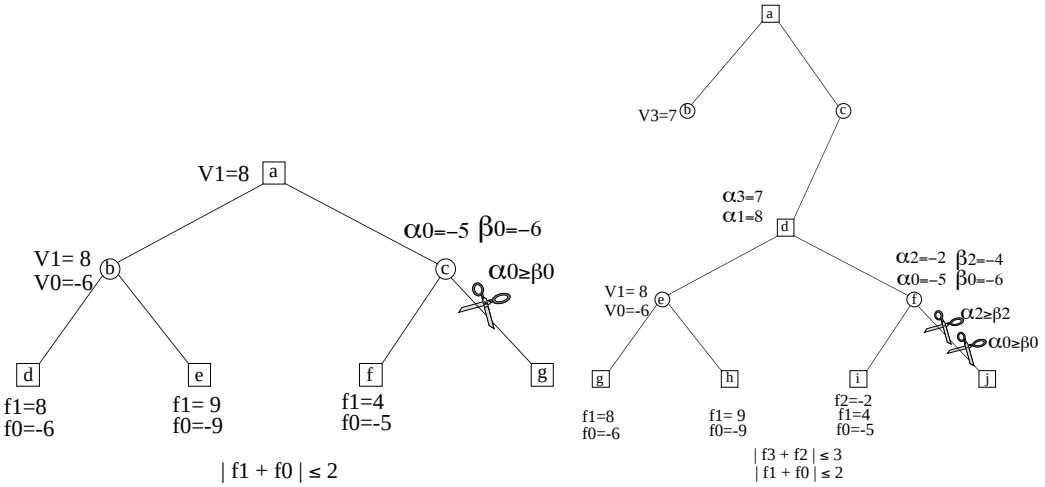

Figure 10: An example of shallow-pruning (left) and deep-pruning (right) performed by $\alpha\beta^*_{1p}$.

3.4.2 Correctness of $\alpha\beta^*_{1p}$

The following theorem shows that $\alpha\beta^*_{1p}$ returns the M^n value of the tree.

Theorem 2 *Let P^n be an n-level player. Let $B_n, \ldots, B_1$ be non-negative numbers such that for every game position s and $n \geq j \geq 1$, $|f_j(s) + f_{j-1}(s)| \leq B_j$. Then,*

$$\alpha\beta^*_{1p}(s, d, P^n, \langle -\infty, \ldots, -\infty \rangle, \langle +\infty, \ldots, +\infty \rangle) = M^*_{1p}(s, d, P^n) = M^n_{(\langle f_n, \ldots, f_0 \rangle, d)}(s)$$

Proof. Since the only difference between $\alpha\beta^*_{1p}$ and M^*_{1p} is that the first does not examine certain nodes, it is sufficient to show that any node pruned by $\alpha\beta^*_{1p}$ can have no effect on the M^n values of its ancestors. Therefore, it cannot change the M^n value of the tree.

The algorithm handles lower and upper bounds for every model. These cutoff bounds are inherited from the upper and lower bounds at the parent node. If model $j+1$ has decided to prune at the parent node, i.e., $\alpha_{j+1} \geq \beta_{j+1}$, it propagates these cutoff values for model j. Otherwise, the algorithm uses the same mechanism as in $\alpha\beta^*$, $\alpha_j = -B^d_{j+1} - \beta_{j+1}$, and $\beta_j = B^d_{j+1} - \alpha_{j+1}$. In addition, the lower bounds for all active models are maximized whenever the search returns the values of one of the successors.

Assume that s is at depth d and all active models agree to prune. Looking at active model j, there are two possibilities shown in Figure 11:

1. At the beginning of the search at node s, $\alpha_j < \beta_j$, and after evaluating one of the successors, α_j is modified to become bigger than β_j. The left part of figure 11 demonstrates the relevant cutoffs at this point of the search. We will show that node s cannot affect the M^{j+1} value of its parent p. If $v_j(s) \geq \alpha_j \geq \beta_j = B^d_{j+1} - \alpha^p_{j+1}$, then $v_{j+1}(p) \geq \alpha^p_{j+1} \geq B^d_{j+1} - v_j(s) \geq v_{j+1}(s)$ and node s cannot affect the M^{j+1} value of p.
2. At the beginning of the search at node s, $\alpha_j \geq \beta_j$. Hence $\alpha^p_{j+1} \geq \beta^p_{j+1}$ at the parent node p. We can continue climbing through the path from node s to the root until we reach the first node k in which $\alpha_{j+l} \geq \beta_{j+l}$ and for its parent node $\alpha_{j+l+1} < \beta_{j+l+1}$, (the existence of such a node is guaranteed since otherwise the search would have not reached node s). The right part of Figure 11 shows the tree and the relevant cutoff values. Note that model $j+l$ decides to prune at node k after evaluating one of its successors but the search has continued and reached node s since other models have not made their decision yet. From the point of view of model $j+l$, all searches of all lower models are redundant and node s cannot affect the M_{j+l} value of node k. Therefore node s can be pruned from the point of view of model j.

Since all active models agree to prune, node s cannot affect the root value and can be pruned. □

We have already mentioned that $\alpha\beta^*$ reduces to $\alpha\beta$ when all the sum-bounds are zero. The same phenomenon occurs for $\alpha\beta^*_{1p}$. When all the sum-bounds are zero the n-level player is reduced to $P^n = (f_n, (-f_n, \ldots, \perp)\ldots)$. The algorithm performs in parallel n identical $\alpha\beta$ searches since all lower bounds are equal and so are all the upper bounds. $\alpha\beta^*_{1p}$ handles the cutoff values exactly as $\alpha\beta$ handles its own in the NEG-MAX version. Therefore, $\alpha\beta^*_{1p}$ evaluates exactly the same positions as $\alpha\beta$ does.

Corollary 2 *For zero sum-bounds $\alpha\beta^*_{1p}$ reduces to $\alpha\beta$.*

3.4.3 Optimality of $\alpha\beta^*_{1p}$

Can we do better than that? The next theorem shows the optimality of $\alpha\beta^*_{1p}$ in the sense that any other directional algorithm that searches for the M^n value of a game-tree must examine any leaf examined by $\alpha\beta^*_{1p}$.

Theorem 3 *Every directional algorithm that computes the M^n value of a game tree with positive sum-bounds must evaluate every terminal node evaluated by $\alpha\beta^*_{1p}$under the same ordering of the search.*

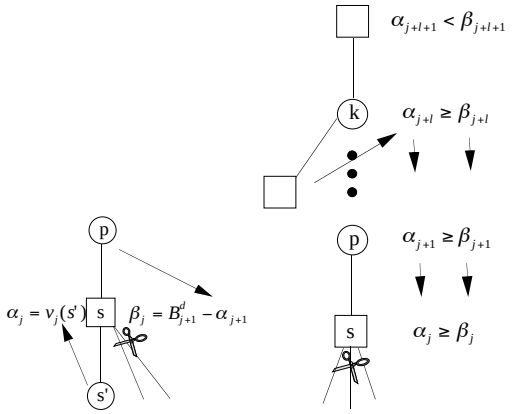

Figure 11: The two possibilities when model j agrees to prune: (Left:) At the beginning of the search at node s, $\alpha_j < \beta_j$, and after evaluating one of the successors, α_j is modified to become bigger than β_j. (Right:) At the beginning of the search at node s, $\alpha_j \geq \beta_j$.

Proof. To prove this theorem we use a method similar to the one used by Korf [15] for showing the optimality of $\alpha\beta$ pruning in a multi-player search. Assume that A is a directional algorithm for computing M^n, and k is a leaf node in the search-tree T, such that k is examined by $\alpha\beta^*_{1p}$ and not examined by A when both algorithms search for the M^n value of T under the same order of search. Consider the state of $\alpha\beta^*_{1p}$ just before evaluating node k. It consists of lower and upper bounds, α_j, β_j, for each model j, and since node k is not pruned, there is an active model i at the parent of node k such that $\alpha_i < \beta_i$. Let us construct a new game-tree, T', by removing all the branches found to the right of the path from the root to k, making node k the last frontier right leaf of T'. Figure 12 illustrates the search-tree and the relevant bounds. The core of the proof is based on the claim that whenever an active model j vetos a pruning decision at a given node, the search under this node must be continued. We will show that by a careful assignment of values to node k it can determine, or affect, the M^n value of T' and therefore cannot be pruned by any directional algorithm.

Let d be the depth of node k. To make the proof more comprehensible we first prove the theorem for shallow trees. For $d = 2$, if node k is not pruned by $\alpha\beta^*$, then $\alpha_{n-1} < \beta_{n-1}$ at node s. We shall assign $\alpha_{n-1} < v_{n-1}(k) < \beta_{n-1}$ and $v_n(k) = B^{d-2}_n - v_{n-1}(k)$ to node k. This assignment propagates up in the tree, $v_{n-1}(s) \leftarrow v_{n-1}(k)$ and $v_n(s) \leftarrow v_n(k)$. It is easy to show that

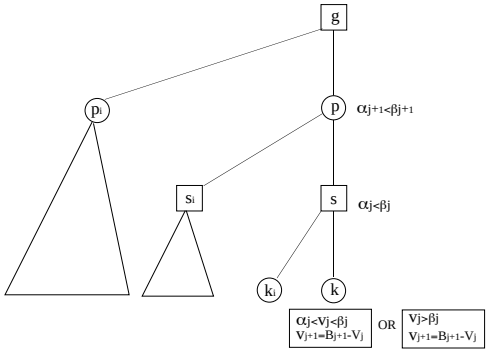

Figure 12: An illustration for the proof of the optimality of $\alpha\beta^*_{1p}$. Every directional algorithm must examine node k. Each of the two alternative assignments affects differently the value of g.

$v_n(k) > \alpha_n$ at the root p. Since α_n at node p is the maximal M^n value of all the subtrees searched so far, node k determines the M^n value of the root and must be evaluated by any search algorithm.

For $d = 3$, if node k is not pruned, then $\alpha_{n-2} < \beta_{n-2}$ at node s. In this case we will show two different assignments to node k that affect the selection of model $n-1$ at node p and therefore affect the M^n value of the root. First, assign $\alpha_{n-2} < v_{n-2}(k) < \beta_{n-2}$ and $v_{n-1}(k) = B^{d-3}_{n-1} - v_{n-2}(k)$ to node k. A similar analysis to the previous case shows that node k determines the M^{n-1} value of node p, $v_{n-1}(p) \leftarrow v_{n-1}(k)$. Second, assign $v_{n-2}(k) > \beta_{n-2}$ and $v_{n-1}(k) = B^{d-3}_{n-1} - v_{n-2}(k)$. In this case node s_i determines the M^{n-1} value of node p. It follows that the selection of model $n-1$ at node p cannot be determined without evaluating node k. The M^n value of node p is determined according to the selection of model $n-1$, and since $\alpha_{n-1} < \beta_{n-1}$ at node p, it has a direct effect on the M^n value of the root and node k cannot be pruned.

Similar arguments hold for deeper nodes in the tree. Let j be the higher model with $\alpha_j < \beta_j$ at node s. If $j = n-1$ we have already shown that node k can determine the M^n value of its grandparent. Since this value is bigger than α_n, node k determines the M^n value of the tree and cannot be pruned. If $j = n-2$, node k can determine the selection of model $n-1$ at its grandparent and therefore the M^n value of its great grandparent. This value cannot be determined without evaluating node k. The influence of node k climbs through the path up to the root and therefore node k cannot be pruned.

If $j < n-2$, we will use the same two different assignments for node k to affect the decision of model $j+2$ at node g and therefore to affect the M^{j+3} value of node g. For the first assignment, let $\alpha_j < v_j(k) < \beta_j$ and $v_{j+1}(k) = B^d_{j+1} - v_j(k)$. In this case, $\alpha_{j+1} < v_{j+1}(k)$ at node p, and therefore node s determines the M^{j+2} of node p. For the second assignment, let $v_j(k) > \beta_j$ and $v_{j+1}(k) = B^d_{j+1} - v_j(k)$. In this case node s_i determines the M^{j+2} of node p. The decision of model $j+2$ at node g is determined according to $v_{j+2}(p)$ which depends on node k and therefore affects $v_{j+3}(g)$. The chain of influence of node k on the decisions of higher models continues climbing in the tree up to the root. Therefore, node k must be evaluated by any search algorithm.

Since algorithm A searches the game-trees T and T' exactly the same, A must also prune node k in T' in contradiction to the fact that node k can determine or affect the M^n value of T'. Therefore, node k must be examined by any directional algorithm. □

4 Average Case Performance - Experimental Study

To study the average performance of the pruning algorithms we conducted a set of experiments using random trees. To build a random uniform tree for a given branching factor b, depth d, modeling level n, and a sum-bound B, we assign a vector of $n+1$ random values to every leaf s such that for any model j, $|v_j(s)+v_{j-1}(s)| \leq B$. (We use the same bound for all models.) The values are randomly drawn from a uniform distribution over a given interval $[-P, P]$ ([-10000,10000] for the following experiments). The dependent variable used for the experiments is the effective branching factor (EBF), $\sqrt[d]{L}$, where L is the number of leaf evaluations. We look at four independent variables: the sum-bound B (to make the number more meaningful we show it as a percentage of P), the depth d, the branching factor b and the modeling level n.

In each of the following experiments, we measured the EBF of $\alpha\beta^*$ and $\alpha\beta^*_{1p}$ as a function of the independent variables by running each algorithm on 100 random trees and averaging the results.

Figure 13 shows the average EBF of both algorithms as a function of the sum-bound for various modeling levels. The branching factor is 4 and the depth is 10. For $ML = 0$, both algorithms traverse the trees exactly like $\alpha\beta$ and the EBF does not depend on the sum-bound. For non-zero levels, the

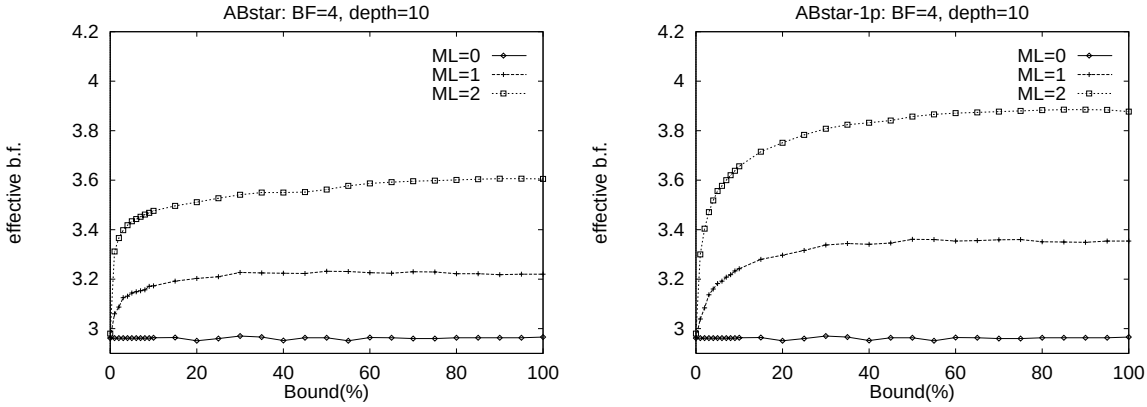

Figure 13: The average EBF of $\alpha\beta^*$(left) and $\alpha\beta^*_{1p}$ (right) as a function of the sum-bound.

EBF increases with the increase of the sum-bound and the increase of the modeling level. Note that due to simple pruning, the average EBF of both algorithms converges to numbers smaller than the branching factors of the non-pruning versions (by Lemma 1 $b^* = \phi_4 = 4.83$ for M^*, and $b^* = 4$ for M^*_{1p}). For infinite sum-bounds (100%), $\alpha\beta^*$ performs only simple pruning and the EBF converges to 3.25 for ML=1, and 3.6 for ML=2, which is much less than the theoretical upper bounds according to Lemma 2, $\sqrt[10]{(\frac{4}{3})4^{10}} = 4.1$, and $\sqrt[10]{(\frac{4}{3})^2 4^{10}} = 4.24$, respectively.

Note also that $\alpha\beta^*$ prunes much more then $\alpha\beta^*_{1p}$. We hypothesize that this phenomenon is a result of $\alpha\beta^*$ being a non-directional algorithm. $\alpha\beta^*_{1p}$simulates the searches of all the models in parallel. Therefore, pruning decisions must be agreed by consensus. $\alpha\beta^*$ performs the searches serially, allowing more elaborate pruning due to the information gathered in previous searches.

Figure 14 shows the average EBF of both algorithms as a function of the search depth for various sum-bounds. The modeling level is 1 and the branching factor is 2. For a zero sum-bound, both algorithms reduce to $\alpha\beta$. For a non-zero sum-bound $\alpha\beta$ has an advantage over $\alpha\beta^*$. The graphs indicate that this gap increases with depth for $\alpha\beta^*_{1p}$, but slightly decreases for $\alpha\beta^*$. Note that the slopes of the graphs emphasize the better performance of $\alpha\beta^*$ over $\alpha\beta^*_{1p}$ for deeper trees.

Figure 15 shows the average EBF of both algorithms as a function of the branching factor for various modeling levels. The sum-bound is 5% and

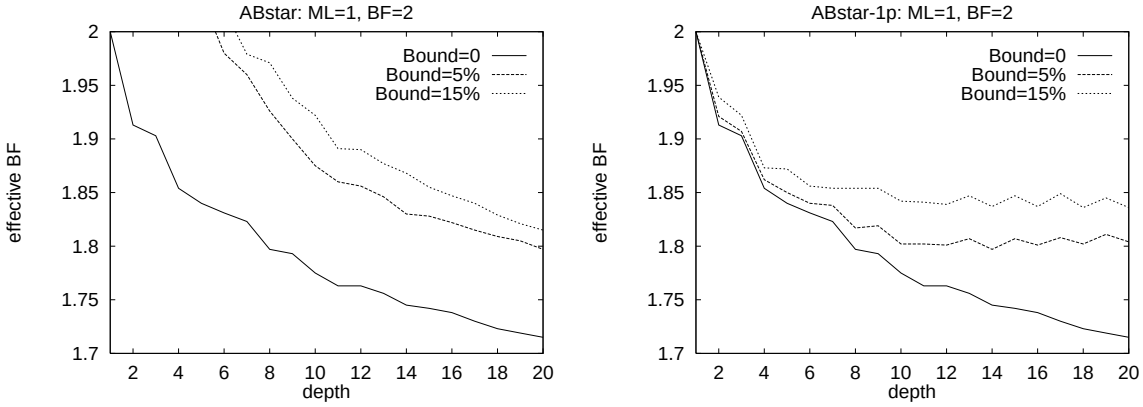

Figure 14: The average EBF of $\alpha\beta^*$(left) and $\alpha\beta^*_{1p}$ (right) as a function of the search depth.

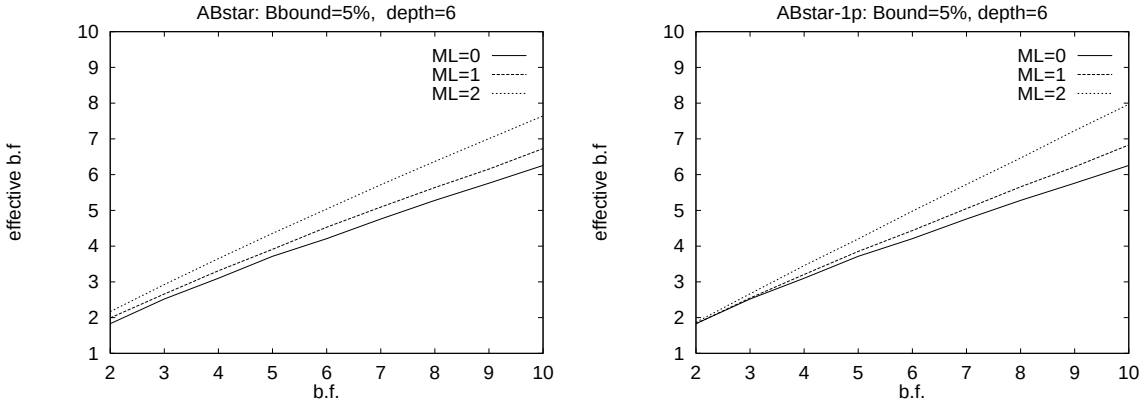

Figure 15: The average EBF of $\alpha\beta^*$(left) and $\alpha\beta^*_{1p}$ (right) as a function of the branching factor.

the depth is 6. These results also emphasize the better performance of $\alpha\beta^*$ over $\alpha\beta^*_{1p}$for large trees. The graphs show that the gap between $\alpha\beta$ and $\alpha\beta^*$ slowly increases with the branching factor.

Simulations with artificial game trees can only be regarded as approximations to the performance of the algorithms in real-life game trees [18]. The next section considers practical issues regarding multi-model search in real game-playing programs.

5 Practical Issues in Multi-model Pruning

There are several issues that should be considered when trying to apply the theoretical multi-model framework to practical game-playing programs. The most important issue is the acquisition of the opponent model. If the playing style of the opponent is known in advance, an appropriate evaluation function that models this style can be constructed. In chess, for example, if it is known that the opponent plays defensively, the user will increase the weight of the "king defense" feature in the model's evaluation function. If past games of the opponent are available, such a process can be automated using book-learning techniques [5, 22, 20]. In previous work we presented a learning algorithm that infers an opponent model based on examples of its past decisions [2]. The algorithm generates a large set of inequalities, expressing the opponent preference of its selected moves over their alternatives, and uses linear programming to solve this set of inequalities. The current version of the learning algorithm is only useful for a one-level player. It is an interesting research problem to generalize it for learning recursive models. However, we expect that one-level model will be sufficient for most practical purposes.

Book-learning algorithms assume that the opponent is stationary. Learning a non-stationary opponent is a manifestation of adaptation to changing environment. In general, the question of learning a non-stationary opponent is difficult. When the opponent does not change its strategy too often, book-learning algorithms can still be applied using simple windowing.

Another issue that should be considered is the acquisition of the bounds on the absolute sum of the player's functions. We assume that the player's and the model's evaluation functions are given and we look for a bound on their absolute sum for any game position. Many practical applications use evaluation functions that are linear combinations of a given set of board terms, $(a_1, \ldots, a_k)$, such as material advantage, mobility, center control etc.

$$f(b) = \sum_{j=1}^{k} w^j a_j$$

When the player's and model's evaluation functions are weighted sums over the same set of terms, it is possible to analytically compute a non-trivial bound over their sum. The sum of such two functions, $f_1(b), f_0(b)$, is:

$$f_1(b) + f_0(b) = \sum_{j=1}^{k} (w_1^j + w_0^j) a_j$$

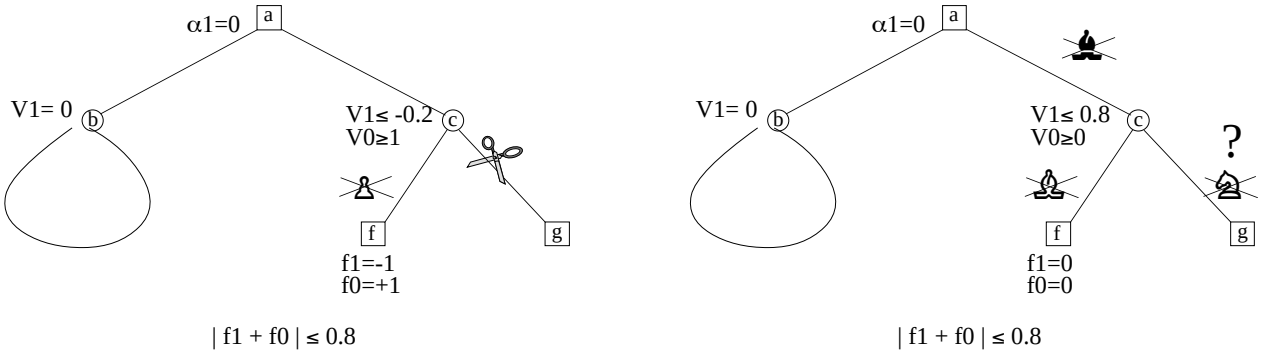

Figure 16: An example of a practical application of $\alpha\beta^*$. The player's function prefers bishops over knights and weighs them by 3.1 and 2.9 respectively. The model's function prefers knights over bishops and weighs them by 3.1 and 2.9 respectively. The sum of the two functions is bounded by 0.8. In the left tree, $\alpha\beta^*$ prunes node g just as $\alpha\beta$ would, since node b is worth at most -0.2 for the player. In the right tree, $\alpha\beta$ with f_1 would prune node g. $\alpha\beta^*$ would not prune this node since node b has an upper bound of 0.8. For example, if the move leading to node g is a knight capture, the model will prefer node g (with a value of 0.2). In such a case node g and therefore node c is worth 0.2 also for the player and determines the M^n value of the tree.

The difference between the two functions is reflected by different weights for some of the board terms. Terms that are weighted symmetrically by the two functions do not affect the sum since $w_1^j + w_0^j = 0$. For two non-symmetric functions (that are not summed to zero), at least one term is not weighted symmetrically, i.e., $w_1^j + w_0^j \neq 0$.

For example. let us look at two simple evaluation functions for chess, consisting only of material advantage. The two functions use the common weights for most pieces: 9 for a queen, 5 for a rook and 1 for a pawn. They only differ in the weights for the bishop and the knight. f_1 weighs bishops by 3.1 and knights by 2.9. f_0 does the opposite – 2.9 for bishops and 3.1 for knights. The absolute sum of these functions is

$$\begin{aligned} |f_1(b) + f_0(b)| &= |(3.1 - 2.9)\text{bishop-advantage} + (2.9 - 3.1)\text{knight-advantage}| \\ &= 0.2\,|\text{bishop-advantage} + \text{knight-advantage}|\,. \end{aligned}$$

Since the maximum material advantage for both terms is 2, a sum-bound for these functions can be obtained by $0.2(2 + 2) = 0.8$.

Figure 16 shows two examples of $\alpha\beta^*$ searches using the above functions, f_1 and f_0. In the left tree, the move leads to node f is a pawn capture. $\alpha\beta^*$

prunes node g just as $\alpha\beta$, since node c is worth at least $+1$ for the model and therefore at most -0.2 for the player. This example demonstrates that if the two functions are not radically different, $\alpha\beta^*$ can prune in spite of the non zero-sum property of its functions. Note that if the move leads to node f would have not been a capture move, c would be worth at least 0 to the model and at most 0.8 for the player, hence $\alpha\beta^*$ would not prune node g while $\alpha\beta$ would.

In the right tree, the move leading to node c involves a black bishop capture and the move leading to node f is a white bishop capture. $\alpha\beta$ with f_1 prunes node g. $\alpha\beta^*$ does not prune this node since node c has an upper bound of 0.8. Node c could be worth more than 0 for the player if, for example, the move leading to node g is a knight capture. In this case, the value of node g for the model would be $-2.9(+1) - 3.1(-1) = 0.2$. Therefore, the model would prefer node g (with a value of 0.2) over node f (with a value of 0). Surprisingly, the value of node g for the player would also be $3.1(+1) + 2.9(-1) = 0.2$. Therefore, node g determines the M^n value of the tree and can not be pruned. Note that position g is an example for a non-terminal position where the zero-sum assumption does not hold. The two players prefer exchanging a white knight for a black bishop over a bishop exchange.

The above discussion is appropriate for cases where the two functions use the same terms with different weights. In other cases, if we do not have an analytical way of determining a bound on the sum of functions, we can use sampling methods. Given two functions, we can perform a large number of simulated games and collect statistics about the sum of the two functions for boards searched during these games. This statistics can be used for estimating a bound on the sum of the functions for any desired confidence.

To test the applicability of the multi-model framework to real domains, we conducted some experiments with $\alpha\beta^*$ in the checkers domain. The first experiment tests the amount of pruning of $\alpha\beta^*$ while searching checkers game trees. The functions f_1 and f_0, used by the player and the model, are both weighted sum of six terms taken from Samuel's function [19] (material advantage, mobility, center control etc.), and a seventh term, *Total*, which counts the number of pieces on the board. The two functions weigh the six terms symmetrically ($w_1^j = -w_0^j, j = 1, \ldots, 6$). The seventh term, *Total*, is given the same negative weight by both functions. That means that both functions prefer piece exchange whenever possible.

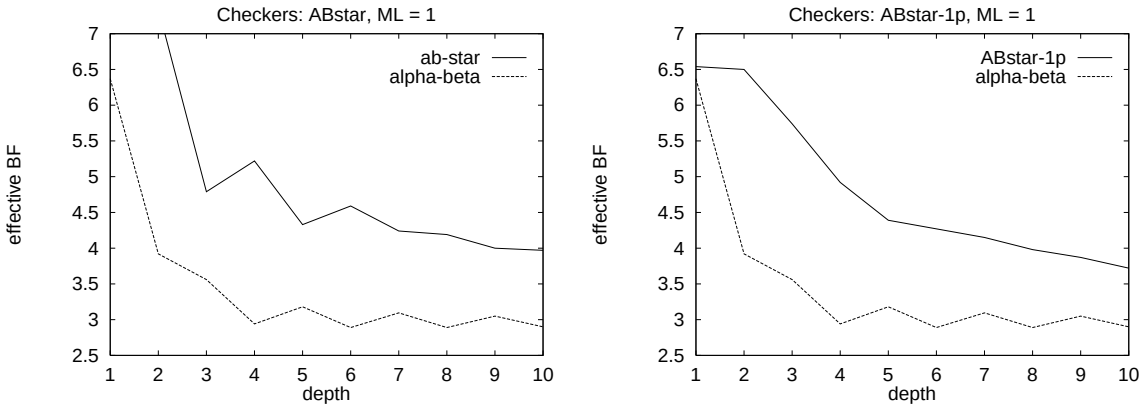

Figure 17: The average EBF of $\alpha\beta^*$and $\alpha\beta^*_{1p}$ as a function of the search depth in the checkers domain.

The bound on the sum of the player's and the model's functions is:

$$|f_1(b) + f_0(b)| = \left|w_1^7 Total(b) + w_0^7 Total(b)\right| \leq \left|(w_1^7 + w_0^7)\right| Total(b)$$

The board at the root of the game tree, r, can be used for computing the sum-bound, since $Total(r)$ is an upper bound on the number of pieces at the leaves of the tree. Moreover, since we use a one-level player in this experiment, the sum-bound does not increase with depth.

Figure 17 shows the average EBF of $\alpha\beta^*$ and $\alpha\beta^*_{1p}$, compared to $\alpha\beta$, as a function of the search depth in the domain of checkers. The results were obtained by counting the number of evaluations per move performed by the algorithms in a tournament of 100 games between them and an $\alpha\beta$ player. $\alpha\beta^*$ and $\alpha\beta^*_{1p}$ use the player $P^1 = (f_1, (f_0, \perp))$, $\alpha\beta$ uses the function f_0.

Note that the performances of both algorithms are similar to their performances for random trees. $\alpha\beta^*$ and $\alpha\beta^*_{1p}$ prune less than $\alpha\beta$ but the average EBF decreases significantly with the search depth.

The last results raise an interesting question. Assume that we allocate a modeling player and an non-modeling opponent the same search resources. Is the benefit achieved by modeling enough to overcome the extra depth that the non-modeling player can search due to better pruning?

To answer this question we have tested iterative deepening version of $\alpha\beta^*$ against a real checkers program based on iterative deepening $\alpha\beta$. Both programs were allocated the same amount of search resources per move. We measured the search resources by the number of leaf evaluations available

Evaluations-per-move	Wins	Draws	Losses	Points	$\alpha\beta^*$ depth	$\alpha\beta$ depth
100	192	646	162	1.030	2.84	3.21
200	287	535	178	1.109	3.24	3.90
300	263	567	170	1.093	3.64	4.24
400	261	566	173	1.088	3.92	4.43
500	210	603	187	1.023	4.11	4.53

Table 1: The results obtained by an iterative deepening version of $\alpha\beta^*$ when played against an iterative deepening version of $\alpha\beta$. Both algorithms were allocated the same search resources – leaf evaluations per move. Each row represents a tournament of 1,000 games. The last two columns show the average search depth of the two algorithms.

for a move. Table 1 shows the results obtained by $\alpha\beta^*$, with the player $P^1 = (f_1, (f_0, \perp))$, against $\alpha\beta$, with the function f_0, for various values of resource limit (leaf evaluations per move). Each row represents the results of a tournament of 1,000 games.

We can see that in all cases $\alpha\beta^*$ searched shallower than $\alpha\beta$ due to its reduced pruning power. However, the deeper search of $\alpha\beta$ was outweighed by the benefit of modeling. When playing against an $\alpha\beta$ player using f_0, $\alpha\beta^*$, using the player $P^1 = (f_1, (f_0, \perp))$, uses the exact opponent function as a model, in contrast to $\alpha\beta$ which wrongly uses $-f_0$ as an opponent model. As a result, while the player prefers exchange positions, $\alpha\beta$ wrongly assumes that it prefers to avoid them.

The reader should note that these results are for a specific game with specific evaluation functions. The tradeoff between the benefit of modeling and the cost of reduced pruning remains open for further research.

6 Conclusions

The minimax approach, which uses a symmetrical information model, has been studied extensively and has reached a state where it is harder to make further significant progress. Korf [14] outlined a general direction for incorporating opponent models into adversary search. Iida et al. [7, 8], and Carmel and Markovitch [2], independently developed opponent modeling search algorithms for one-level models. Carmel and Markovitch [4] generalized the framework for multi-model search.

One of the reasons for the tremendous success of minimax is the $\alpha\beta$ prun-

ing technique. Opponent modeling search requires similar pruning methods to become competitive. Korf [13] raised doubts whether pruning is possible in opponent modeling search. Iida [9] describe the β-pruning algorithm for one-level search. This method is restricted to the pruning performed by the $\alpha\beta$ simulation of the opponent. The simple pruning algorithm described in Section 3.1 is a generalization of this technique to multi-model search.

This paper presents a general framework for pruning in multi-model search. We show that pruning is impossible without further restrictions. We prove a sufficient condition for pruning based on a bound on the sum of the player's and the model's evaluation functions. This restriction is a similar to the one used by Korf [15] for multi-player search. We then present two algorithms that utilize this restriction for pruning. We prove correctness and optimality and study theoretically and experimentally the complexity of the algorithms.

The pruning power of the algorithms depends on the sum-bounds. We conducted a set of experiments in artificial random trees and in the checkers domain. The results indicate that for small sum-bounds, and a player with modeling-level one, the pruning achieved by $\alpha\beta^*$ is significant and close to that achieved by $\alpha\beta$.

Berliner and McConnell [1] deal with the difficulty in finding optimistic and pessimistic bounds on the real values of game-tree positions. However, their results are not applicable for multi-model search where we look for bounds on the *sum* of two functions. The sum-bounds for a multi-model search reflect the maximal possible difference between the subjective evaluations done by the player and its opponent model. In Section 5 we presented practical methods for computing such bounds.

The question whether the benefit of modeling outweighs the cost of reduced pruning depends on the domain and on the evaluation function used. In this work we showed an example, in the checkers domain, where the benefit of modeling outweighs the cost of reduced pruning power.

The multi-model framework allows the use of arbitrary opponent models. This work points out that pruning is significantly reduced for a model which is radically different from the player's strategy. However, for zero-sum games, we do not expect the players to evaluate boards in a radically different way. In such cases, the pruning power of $\alpha\beta^*$ is significant.

References

[1] Hans J. Berliner and Chris McConnell. B* probability based search. *Artificial Intelligence*, 86:97–156, 1996.

[2] David Carmel and Shaul Markovitch. Learning models of opponent's strategies in game playing. In *Proceedings of the AAAI Fall Symposium on Games: Planning and Learning*, pages 140–147, Raleigh, NC, October 1993.

[3] David Carmel and Shaul Markovitch. The M* algorithm: Incorporating opponent models into adversary search. Technical Report CIS9402, Technion, March 1994.

[4] David Carmel and Shaul Markovitch. Incorporating opponent models into adversary search. In *Proceedings of thirteenth National Conference on Artificial Intelligence (AAAI 96)*, pages 120 – 125, Portland, Oregon, August 1996.

[5] J. Christensen and R. E. Korf. A unified theory of heuristic evaluation functions and its application to learning. *Proceedings of the fifth National Conference on Artificial Intelligence*, 1986.

[6] P. J. Gmytrasiewicz, E. H. Durfee, and D. K. Wehe. A decision theoretic approach to coordinating multiagent interactions. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI 91)*, pages 62 – 68, Sydney Australia, August 1991.

[7] H. Iida, Jos W. H. M. Uiterwijk, H. J. van den Herik, and I. S. Herschberg. Potential applications of opponent-model search, part I: The domain of applicability. *ICCA Journal*, 16(4):201–208, 1993.

[8] H. Iida, Jos W. H. M. Uiterwijk, H. J. van den Herik, and I. S. Herschberg. Potential applications of opponent-model search, part II: Risks and strategies. *ICCA Journal*, 17(1):10–14, Mar 1994.

[9] Hiroyuki Iida. *Heuristic Theories on Game-Tree Search.* PhD thesis, Tokyo University, 1994.

[10] Peter J. Jansen. Problematic positions and speculative play. In T. A. Marsland and J. Schaeffer, editors, *Computers, Chess and Cognition*, pages 169–182. Springer New York, 1990.

[11] Peter J. Jansen. *Using Knowledge about the Opponent in Game-tree Search*. PhD thesis, Carnegie Mellon University, 1992.

[12] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6, No. 4:293–326, 1975.

[13] R. E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27:97–109, 1985.

[14] Richard E. Korf. Generalized game trees. In *Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI 89)*, pages 328–333, Detroit, MI, August 1989.

[15] Richard E. Korf. Multi-player alpha-beta pruning. *Artificial Intelligence*, 48:99–111, 1991.

[16] C. A. Luckhardt and K. B. Irani. An algorithmic solution of n-person games. In *Proceeding of the Ninth National Conference on Artificial Intelligence (AAAI-86)*, pages 158–162, Philadelfia, PA, August 1986.

[17] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley, 1984.

[18] Aske Plaat. *Research RE:search & RE-search*. PhD thesis, Erasmus University, 1996.

[19] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal*, 3:211–229, 1959.

[20] Arthur L. Samuel. Some studies in machine learning using the game of checkers II–recent progress. *IBM Journal*, 11:601–617, 1967.

[21] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[22] M. van der Meulen. Weight assessment in evaluation functions. In D. F. Beal, editor, *Advances in Computer Chess 5*, pages 81–89. Elsevier Science Publishers, Amsterdam, 1989.