Incorporating Opponent Models into 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

This work presents a generalized theoretical framework that allows incorporation of opponent models into adversary search. We present the M^* algorithm, a generalization of minimax that uses an arbitrary opponent model to simulate the opponent's search. The opponent model is a recursive structure consisting of the opponent's evaluation function and its model of the player. We demonstrate experimentally the potential benefit of using an opponent model. Pruning in M^* is impossible in the general case. We prove a sufficient condition for pruning and present the $\alpha\beta^*$ algorithm which returns the M^* value of a tree while searching only necessary branches.

Introduction

The minimax algorithm (?) has served as the basic decision procedure 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 some external source. How can such knowledge be exploited? In game theory, the question is known as the *best response* problem - looking for an optimal response to a given opponent model (?). However, there is almost no theoretical work on using opponent models in adversary search. Korf (?) outlined a method of utilizing multiple-level models of evaluation functions. Carmel and Markovitch (?) developed an algorithm for utilizing an opponent model defined by an evaluation function and a depth of search.

Jansen (?) describes two situations where it is important to consider the opponent's strategy. One is a *swindle* position, where the player has reason to believe that the opponent will underestimate a good move, and will therefore play a poorer move instead. Another situation is a *trap* position, where the player expects the opponent to overestimate and therefore play a bad move. Another situation, where an opponent model can be beneficial, is a losing position (?). When all possible moves lead to a loss, an opponent model can be used to select a swindle move.

The goal of this research is to develop a theoretical framework that allows exploitation of an opponent model. We start by defining an opponent model to be a recursive structure consisting of a utility function of the opponent and the player model (held by the opponent). We then describe two algorithms, M^* and M^*_{1-pass}, generalizations of minimax that can use an opponent model. Pruning in M^* is not possible in the general case. However, we have developed the $\alpha\beta^*$ algorithm that allows pruning given certain constraints over the relationship between the player's and the model's strategies.

The M^* algorithm

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

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

The algorithm selects a move in the following way. It generates the successors of the current position. It then applies φ on each of the successors to obtain the opponent's response and evaluates each of the resulting states by applying the algorithm recursively (with reduced depth limit). It then selects the successor with the highest value. The algorithm returns the static evaluation of the input position when the depth limit is zero.

Note that M returns a *value*. M can also be defined to return the *state* with the maximal value instead of the value itself. To simplify the discussion, from now on we will assume that M returns either the state or the value according to the context.

Let $M^0_{(\langle f_0\rangle,d)}$ be the regular minimax algorithm that searches to depth d using an evaluation function f_0. Minimax uses itself with $d-1$ and $-f_0$ as an opponent model, therefore it can be defined as a special case of M:

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

Assume that the player uses an evaluation function f_1. A natural candidate to serve as the model φ in the M algorithm is M^0 with another evaluation function f_0. We call this special case of M the M^1 algorithm:

$$M^1_{(\langle f_1,f_0\rangle,d)}(s) = M(s, d, f_1, M^0_{(\langle f_0\rangle,d-1)}).$$

The M^1 algorithm works by simulating the opponent's minimax search (to depth $d-1$) to find its selected move and evaluates the selected move by calling itself recursively (to depth $d-2$).

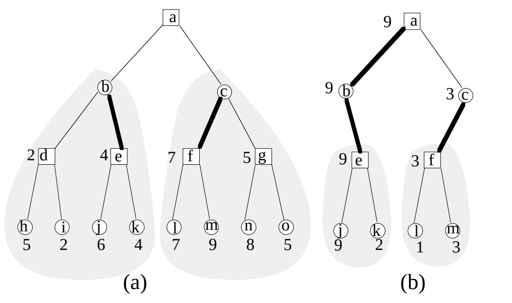

Figure 1: The search trees spanned by calling M^1. Part (a) shows the two calls to minimax for determining the opponent's choice. Part (b) shows the recursive calls to M^1 for evaluating the nodes selected by the opponent.

Figure **??** shows an example of a search to depth 3 performed by the M^1 algorithm. Part *(a)* shows the two calls to minimax to simulate the opponent's search. Note that the opponent is a maximizer. Part *(b)* of the figure shows the two recursive calls to M^1 applied to the boards selected by the opponent. Note that while the opponent's minimax procedure believes that in node e the player selects the move leading to node k, the player actually prefers node j.

We can define the M^n algorithm, for any n, to be the M algorithm with $\varphi = M^{n-1}$. M^n can be formally defined as follows:

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

Thus, a player using the M^1 algorithm assumes that its opponent uses M^0 (minimax), a player using the M^2 algorithm assumes that its opponent uses M^1, etc. We will define the M^* algorithm that includes every M^n algorithm as a special case. M^* receives as a parameter a *player* which includes information about both the player's evaluation function and its model of the opponent.

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

1. *Given an evaluation function f, $P = (f, NIL)$ is a* player *(with a modeling-level 0).*
2. *Given an evaluation function f and a* player *O (with modeling level $n-1$), $P = (f, O)$ is a* player *(with a modeling level n).*

The first element of a player is called the player's strategy *and the second element is called the* opponent model.

Thus, a zero-level modeling player, (f_0, NIL), is one that does not model its opponent. A one-level modeling player, $(f_1, (f_0, NIL))$, is one that has a model of its opponent, but assumes that its opponent is a zero-level modeling player. A two-level modeling player, $(f_2, (f_1, (f_0, NIL)))$, is one that uses a strategy f_2, and has a model of its opponent, $(f_1, (f_0, NIL))$. The opponent's model uses a strategy f_1 and has a model, (f_0, NIL), of the player. The recursive definition of a player is in the spirit of the Recursive Modeling Method by Gmytrasiewicz, Durfee and Wehe (**?**).

M^* receives a position, a depth limit, and a *player*, and outputs a move selected by the player and its value. The algorithm generates the successor boards and simulates the opponent's search from each of them 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 optional moves by evaluating the outcome of its opponent's reaction by applying the algorithm recursively using its own strategy.

Figure **??** shows an example of a search tree spanned by $M^*(a, 3, f_2(f_1, f_0))$. The numbers at the bottom are the static values of the leaves. The recursive calls applied to each node are listed next to the node. The dashed lines indicate which move is selected by each recursive call.

The player simulates its opponent's search from nodes b and c. The opponent simulates the player by using its own model of the player from nodes d and e.

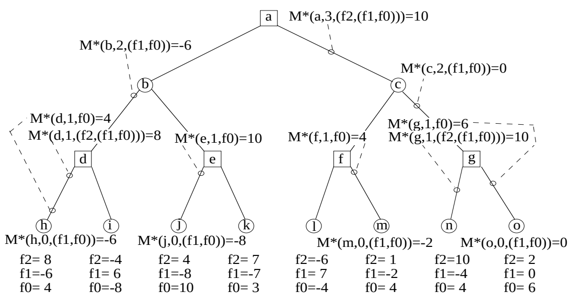

Figure 2: The set of recursive calls generated by calling $M^*(a, 3, (f_2(f_1, f_0)))$. Each call is written next to the node it is called from. The dashed lines show which move is selected by each call.

At node d the model of the player used by the opponent (f_0) selects node h. The opponent then applies its f_1 function to node h and concludes that node h, and therefore node d, are worth -6. The opponent then applies the player's model (f_0) to node e, concludes that the player select node j, applies its own function (f_1) to node j, and decides that node j, and therefore node e, are worth -8. Therefore, the opponent model, when applied to node b, selects the move that leads to node d. The player then evaluates node d using its own criterion (f_2). It applies M^* to node d and concludes that node d, and therefore node b, are worth 8. Simulation of the opponent from node c yields the selection of node g. The player then evaluates g according to its own strategy and finds that it is worth 10 (the value of node n). Note that when the opponent simulates the player from node g, it wrongly assumes that the player selects node o. Therefore, the player selects the move that leads to c with a value of 10. Note that using a regular minimax search with f_2 would have resulted in selecting the move that leads to node b with a value of 7. The formal listing of the M^* algorithm is shown in figure **??**.

```
Procedure $M^*$ $(pos, d, (f_{pl}, O))$
 if $d = 0$ then return $\langle NIL, f_{pl}(pos) \rangle$
 else
  $max\_value \leftarrow -\infty$
  $S \leftarrow \sigma(pos)$
  for each $s \in S$
   if $d = 1$ then $pl\_v \leftarrow f_{pl}(s)$
   else
    $\langle op\_b, op\_v \rangle \leftarrow M^*\ (s, d-1, O)$
    $\langle pl\_b, pl\_v \rangle \leftarrow M^*\ (op\_b, d-2, (f_{pl}, O))$
   if $pl\_v > max\_value$
    $max\_value \leftarrow pl\_v$
    $max\_board \leftarrow s$
  return $\langle max\_board, max\_value \rangle$
```

Figure 3: The M^* algorithm

The M^n algorithm calls M^0 when n reaches 0. However, note that M^* does not contain such a call. This will work correctly when the modeling level is larger than the search depth. If the modeling level of the original player n is smaller than the depth of the search tree d, we replace the 0-level modeling player f_0 by $\overbrace{(f_0, (-f_0, (f_0, \ldots)))}^{d-n}$.

A one-pass version of M^*

It is obvious that the M^* algorithm performs multiple expansions of parts of the search tree. We have developed another version of the M^* algorithm, called M^*_{1-pass}, that expands the tree one time only, just as minimax does. The algorithm expands the search tree in the same manner as minimax. However, node values are propagated differently. Whereas minimax propagates only one value, M^* propagates $n+1$ values, $(V_n, \ldots, V_0)$. The value V_i represents the merit of the current node according to the i-level model, f_i. M^*_{1-pass} passes values associated with the player and values associated with the opponent in a different manner. In a player's node (a node where it is the player's turn to play)[1], for values associated with the player $(V_n, V_{n-2}, \ldots)$, V_i receives the maximal V_i value among its children. For values associated with the opponent $(V_{n-1}, V_{n-3}, \ldots)$, V_i receives the V_i value of the child that gave the maximal value to V_{i-1}. For example, the opponent believes (according to the model) that the player evaluates nodes by V_{n-2}. At a player's node, the opponent assumes that the player will select the child with maximal V_{n-2} value. Therefore, the value of the current node for the opponent is the V_{n-1} value of the selected child with the maximal V_{n-2} value. At an opponent's node, we do the same but the roles of the opponent and the player are switched.

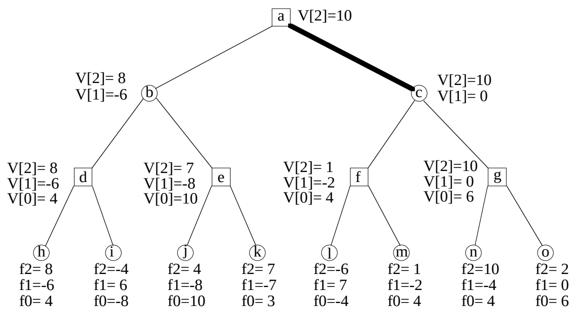

Figure 4: The value vectors propagated by M^*_{1-pass}. This is the same tree as the one shown in Figure **??**.

Figure **??** shows an example for a tree spanned by M^*_{1-pass}. This is the same tree as the one shown in

[1]Traditionally such a node is called a MAX node. However, we assume that both players are maximizers.

Figure **??**. Let us look at node e to understand the way in which the algorithm works. Three players 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 $f0$ and selects board 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 j, selected by the player model, using its own evaluation function $f1$, ($f1(j) = -8$). The player actually prefers node k with the higher $f2$ value ($f2(k) = 7$). Thus, the vector propagated from node e is $\langle 7, -8, 10\rangle$. Note that the values in the vectors correspond to the results of the recursive calls in figure **??**. Figure **??** lists the M^*_{1-pass} algorithm.

```
Procedure $M^*_{1-pass}$ $(pos, d, (f_n, (f_{n-1}, (\ldots, f_0)\ldots)))$
 if $d = 0$ then return $\langle f_n(pos), \ldots, f_0(pos)\rangle$
 else $S \leftarrow \sigma(pos)$
  $V \leftarrow \langle -\infty, \ldots, -\infty\rangle$
  for each $s \in S$
   succ_V $\leftarrow M^*_{1-pass}$ $(s, d-1, (f_n, (f_{n-1}, \ldots, f_0)\ldots))$
   for each $i$ associated with current player
    if succ_V$[i] > V[i]$ then
     $V[i] \leftarrow$ succ_V$[i]$
     if $i < n$ then $V[i+1] \leftarrow$ succ_V$[i+1]$
  return $\langle V[n], \ldots, V[d]\rangle$
```

Figure 5: M^*_{1-pass}: A version of the M^* algorithm that performs only one pass over the search tree

Properties of M^*

The following theorem shows that M^* and M^*_{1-pass} return the same value.

Theorem 1 *Assume that P is a n-level modeling player. Let $\langle v, b\rangle = M^*(pos, d, P)$, and let $\langle V[n]\rangle = M^*_{1-pass}(pos, d, P)$. Then $v = V[n]$.*

The proof for all the theorems in this paper can be found in (**?**).

It seems as though M^*_{1-pass} is always prefered over M^* because it expands less nodes. M^*_{1-pass} expands each node in the tree only once, while M^* re-expands many nodes. However, while M^*_{1-pass} performs less *expansions* than M^*, it may perform more *evaluations*. An upper limit on the number of node expansions and evaluations in M^* and M^*_{1-pass} is given in the following theorem.

Theorem 2 *Assume that M^* and M^*_{1-pass} search a tree with a uniform branching factor b and depth d.*

1. *The number of calls to the expansion function by M^* is bounded by $(b+1)^{d-1}$. The number of calls by M^*_{1-pass} is $\frac{b^d-1}{b-1}$.*
2. *The number of calls to evaluation functions by M^* is bounded by $(b+1)^d$. the number of calls by M^*_{1-pass} is db^d.*

The theorem implies that M^* and M^*_{1-pass} each has an advantage. If it is more important to reduce the number of node expansions than the number of evaluation function calls then one should use M^*_{1-pass}. Otherwise, M^* should be used. For example, for $d = 10$ and $b = 30$ and a one-level modeling player, M^* expands 1.3 times more nodes than M^*_{1-pass} but M^*_{1-pass} performs 1.5 times more calls to evaluation functions. Note that when the set of evaluation functions consist of the same features (perhaps with different weights), the overhead of multiple evaluation is reduced significantly.

We have already shown that M^* is a generalization of minimax. An interesting property of M^* is that it always selects a move with a value greater or equal to the value of the move selected by minimax that searches the same tree with the same strategy.

Theorem 3 *Assume that M^* and Minimax use the same evaluation function f. Then $Minimax(pos, d, f) \leq M^*(pos, d, (f, O))$ for any opponent model O.*

The theorem states that if you have a *good* reason to believe that your opponent's model is different than yours, you could only benefit by using M^* instead of minimax. The reader should note that the above theorem does not mean that M^* selects a move that is better according to some objective criterion, but rather a subjectively better move (from the player's point of view, according to its strategy). If the player does not have a reliable model of its opponent, then playing minimax is a good cautious strategy.

Adding pruning to M*

One of the most significant extensions of the minimax algorithm is the $\alpha\beta$ pruning technique. Is it possible to add such an extension to M^* as well? Unfortunately, if we assume a total independence between f_1 and f_0, it is easy to show that such a procedure cannot exist. Knowing a lower bound for the opponent's evaluation of a node does not have any implications on the value of the node for the player. A similar situation arises in *MAXN*, a multi-player game tree search algorithm. Luckhardt and Irani (**?**) conclude that pruning is impossible in *MAXN* without further restrictions about the players' evaluation functions. Korf (**?**) showed that a shallow pruning for *MAXN* is possible if we assume an upper bound on the sum of the players' functions, and a lower bound on every player's function.

The basic assumption used for the original $\alpha\beta$ algorithm is that $f_1 + f_0 = 0$ (the zero-sum assumption). This assumption is used to infer a bound on a value of a node for a player based directly on the opponent's value. A natural relaxation to this assumption is $|f_1 + f_0| \leq B$. This assumption means that while f_1 and $-f_0$ may evaluate a board differently, this difference is bounded. For example, the player may prefer a rook over a knight while the opponent prefers the opposite. 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.

The above assumption can be used in the context of the M^*_{1-pass} algorithm to determine a bound on $V_i + V_{i-1}$ at the leaves level. But in order to be able to prune using this assumption, we first need to determine how these bounds are propagated up the search tree.

Lemma 1 *Assume that A is a node in the tree spanned by M^*_{1-pass}. Assume that $S_1, \ldots, S_k$ are its successors. If there exist non-negative bounds $B_0, \ldots, B_n$, such that for each successor S_j, and for each model i, $\left|V_{S_j}[i] + V_{S_j}[i-1]\right| \leq B_i$. Then, for each model $1 \leq i \leq n$, $|V_A[i] + V_A[i-1]| \leq B_i + 2 \cdot B_{i-1}$.*

Based on lemma **??**, we have developed an algorithm, $\alpha\beta^*$, that can perform a shallow and deep pruning assuming bounds on the absolute sum of functions of the player and its opponent model. $\alpha\beta^*$ takes as input a position, a depth limit, and for each model i, a strategy f_i, an upper bound B_i on $|f_i + f_{i-1}|$, and a cutoff value α_i. It returns the M^* value of the root by searching only those nodes that might affect this value.

The algorithm works similarly to the original $\alpha\beta$ algorithm, but is much more restricted as to which subtrees can be pruned. The $\alpha\beta^*$ algorithm only prunes branches that *all* models agree to prune. In regular $\alpha\beta$, the player can use the opponent's value of a node to determine whether it has a chance to attain a value that is better than the current cutoff value. This is based on the opponent's value being exactly the same as the player's value (except for the sign). In $\alpha\beta^*$, the player's function and the opponent's function are not identical, but their difference is bounded. The bound on $V_i + V_{i-1}$ depends on the distance from the leaves level. At the leaves level, it can be directly computed using the input B_i. At distance d, the bound can be computed from the bounds for level $d-1$ as stated by lemma **??**[2].

[2] $\alpha\beta^*$ computes the bound B for each node. However, a table of the B values can be computed once at the beginning of the search, since they depend only on the B_i and the depth.

A cutoff value α_i for a node v is the highest current value of all the ancestors of v from the point of view of player i. α_i is modified at nodes where it is player i's turn to play, and is used for pruning where it is player $i-1$'s turn to play. At each node, for each i associated with the player whose turn it is to play, α_i is maximized any time V_i is modified. For each i such that $i-1$ is associated with the current player, the algorithm checks whether the i player wants its model (the $i-1$ player) to continue its search.

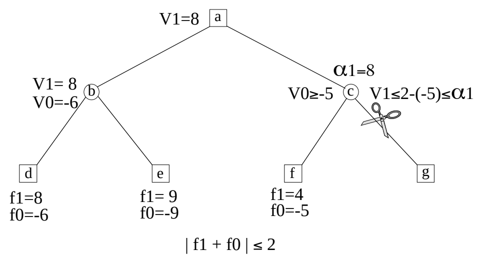

Figure 6: An example of pruning performed by $\alpha\beta^*$.

Figure **??** shows a search tree where every leaf l satisfies the bound constraint $|f_1(l) + f_0(l)| \leq 2$. This bound allows the player to perform a cutoff of branch g, knowing that the value of node c for the opponent will be at least -5. Therefore, its value will be at most 7 for the player.

The $\alpha\beta^*$ algorithm is listed in figure **??**. The following theorem proves that $\alpha\beta^*$ always returns the M^* value of the root.

Theorem 4 *Let $V = \alpha\beta^*(pos, d, P, (-\infty, \ldots, -\infty))$. Let $V' = M^*_{1-pass}(pos, d, P')$ where P' is P without the bounds. Assume that for any leaf l of the game tree spanned from position pos to depth d, $|f_i(l) + f_{i-1}(l)| \leq b_i$. Then, $V = V'$.*

As the player function becomes more similar to its opponent model (but with an opposite sign), the amount of pruning increases up to the point where they use the same function where $\alpha\beta^*$ prunes as much as $\alpha\beta$.

Experimental study: The potential benefit of M^*

We have performed a set of experiments to test the potential merit of the M^* algorithm. The experiments involve three players: *MSTAR*, *MM* and *OP*. *MSTAR* is a one-level modeling player. *MM* and *OP* are regular minimax players (using $\alpha\beta$ pruning). The experiments were conducted using the domain of checkers. Each tournament consisted of 800 games. A limit of 100 moves per game was set during the tournament.

```
Procedure $\alpha\beta^*(pos, d,$
                    $((f_n, b_n)(\ldots, (f_0, b_0))\ldots))(\alpha_n, \ldots, \alpha_0))$
 if $d = 0$ then return $\langle f_n(pos), \ldots, f_0(pos)\rangle$
 else
  $B \leftarrow$ ComputeBounds$(d, \langle b_n, \ldots, b_0\rangle)$
  $V \leftarrow \langle -\infty, \ldots, -\infty\rangle$
  $S \leftarrow \sigma(pos)$
  for each $s \in S$
   succ_V $\leftarrow \alpha\beta^*(s, d-1,$
                    $((f_n, b_n)(\ldots, (f_0, b_0))\ldots))(\alpha_n, \ldots, \alpha_0))$
    loop for each $i$ associated with current player
     if succ_V$[i] > V[i]$ then
      $V[i] \leftarrow$ succ_V$[i]$
      if $i < n$ then $V[i+1] \leftarrow$ succ_V$[i+1]$
     $\alpha_i \leftarrow \max(\alpha_i, V[i])$
    if for every $i$ not associated with current player
     $[\alpha_i \geq B[i] - V[i-1]]$ then return $\langle V[n], \ldots, V[d]\rangle$
 return $\langle V[n], \ldots, V[d]\rangle$

Procedure ComputeBounds$(d, \langle b_n, \ldots, b_0\rangle)$
 if $d = 0$ then return $\langle b_n, \ldots, b_0\rangle$
 else
  succ_B $\leftarrow$ ComputeBounds$(d-1, \langle b_n, \ldots, b_0\rangle)$
  loop for each $i$ associated with current player
   $B[i] \leftarrow succ\_B[i] + 2 \cdot succ\_B[i-1]$
   if $i < n$ then $B[i+1] \leftarrow$ succ_B$[i+1]$
  return B
```

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

In the first experiment all players searched to depth 4. *MM* and M^* used one function while *OP* used another with equivalent power. *MSTAR* knows its opponent's function while *MM* implicitly assumes that its opponent uses the opposite of its own function. Both *MSTAR* and *MM*, limited by their own search depth, wrongly assume that *OP* searches to depth 3. The following table shows the results obtained.

	Wins	Draws	Losses	Points
MM vs. *OP*	94	616	90	804
MSTAR vs. *OP*	126	618	56	870

The first row of the table shows that indeed the two evaluation functions are of equivalent power. The second row shows that *MSTAR* indeed benefited using the extra knowledge about the opponent.

In the second experiment all players searched to depth 4. The three players used the same evaluation function $f(b,pl) = Mat(b,pl) - \frac{1}{25} \cdot Tot(b)$ where $Mat(b,pl)$ returns the material advantage of player pl and $Tot(b)$ is the total number of pieces. However, while M^* knows its opponent's function, *MM* implicitly assumes that its opponent, o, uses the function $f(b,o) = -f(b,pl) = -Mat(b,pl) + \frac{1}{25} \cdot Tot(b) = Mat(b,o) + \frac{1}{25} \cdot Tot(b)$. Therefore, while *OP* prefers exchange positions, *MM* assumes that it prefers to avoid them. The following table shows the results obtained.

	Wins	Draws	Losses	Points
MM vs. *OP*	171	461	168	803
MSTAR vs. *OP*	290	351	159	931

This result is rather surprising. Despite the fact that all players used the same evaluation function, and searched to the same depth, the modeling program achieved a significantly higher score.

We repeated the last experiment replacing the M^*_{1-pass} algorithm with $\alpha\beta^*$and measured the amount of pruning. The bound used for pruning is $|f_1 + f_0| = |Mat(b,pl) - \frac{1}{25} \cdot Tot(b) + Mat(b,o) - \frac{1}{25} \cdot Tot(b)| \leq \frac{2}{25} \cdot Tot(b)$. While the average number of leaves-per-move for a search to depth 4 by M^*_{1-pass} was 723, $\alpha\beta^*$ managed to achieve an average of 190 leaves-per-move. $\alpha\beta$ achieved an average of 66 leaves-per-move.

The last results raise an interesting question. Assume that we allocate a modeling player and an unmodeling 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 the better pruning? We have tested the question in the context of the above experiment. We wrote an iterative deepening versions of both $\alpha\beta$ and $\alpha\beta^*$ and let the two play against each other with the same limit on the number of leaves.

	Wins	Draws	Losses	Points
$\alpha\beta^*$ vs. *OP*	226	328	246	780

This result is rather disappointing. The benefit of modeling was outweighed by the cost of the reduced pruning. In general this is an example of the known tradeoff between knowledge and search. The question of when is it worthwhile to use the modeling approach remains open and depends mostly on the particular nature of the search tree and evaluation functions.

Conclusions

This paper describes a generalized version of the minimax algorithm that can utilize different opponent models. The M^* algorithm simulates the opponent's search to determine its expected decision for the next move, and evaluates the resulted state by searching its associated subtree using its own strategy.

Experiments performed in the domain of checkers demonstrated the advantage of M^* over minimax. Can the benefit of modeling outweigh the cost of reduced pruning? We don't have a conclusive answer for that question. We expect the answer to be dependent on the particular domain and particular evaluation functions used.

There are still many remaining open questions. How does an error in the model effect performance? How can a player use an uncertain model? How can a player acquire a model of its opponent? We tackled these questions but could not include the results here due to lack of space (a full version of the paper that includes these parts is available as (**?**).)

In the introduction we raised the question of utilizing knowledge about the opponent's decision procedure in adversary search. We believe that this work presents a significant progress in our understanding of opponent modeling, and can serve as a basis for further theoretical and experimental work in this area.

References

Berliner, H. 1977. Search and knowledge. In *Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI 77)*, 975–979.

Carmel, D., and Markovitch, S. 1993. Learning models of opponent's strategies in game playing. In *Proceedings of the AAAI Fall Symposium on Games: Planning and Learning*, 140–147.

Carmel, D., and Markovitch, S. 1996. Learning and using opponent models in adversary search. Technical Report CIS report 9609, Technion.

Gilboa, I. 1988. The complexity of computing best response Automata in repeated games. *Journal of economic theory* 45:342 –352.

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

Jansen, P. 1990. Problematic positions and speculative play. In Marsland, T., and Schaeffer, J., eds., *Computers, Chess and Cognition.* Springer New York. 169–182.

Korf, R. E. 1989. Generalized game trees. In *Proceeding of the International Joint Conference on Artificial Intelligence (IJCAI 89)*, 328–333.

Korf, R. E. 1991. Multi-player alpha-beta pruning. *Artificial Intelligence 48, 99-111.*

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

Shannon, C. E. 1950. Programming a computer for playing chess. *Philosophical Magazine, 41, 256-275.*