More Games of No Chance
MSRI Publications
Volume **42**, 2002

Alpha-Beta Pruning Under Partial Orders

MATTHEW L. GINSBERG AND ALAN JAFFRAY

Abstract. Alpha-beta pruning is the algorithm of choice for searching game trees with position values taken from a totally ordered set, such as the set of real numbers. We generalize to game trees with position values taken from a partially ordered set, and prove necessary and sufficient conditions for alpha-beta pruning to be valid. Specifically, we show that shallow pruning is possible if and only if the value set is a lattice, and full alpha-beta pruning is possible if and only if the value set is a distributive lattice. We show that the resulting technique leads to substantial improvements in the speed of algorithms dealing with card play in contract bridge.

1. Introduction

Alpha-beta (α-β) pruning is widely used to reduce the amount of search needed to analyze game trees. However, almost all discussion of α-β in the literature is restricted to game trees with real or integer valued positions. It may be useful to consider game trees with other valuation schema, such as vectors, sets, or constraints. In this paper, we attempt to find the most general conditions on a value set under which α-β may be used.

The intuition underlying α-β is that it is possible to eliminate from consideration portions of the game tree that can be shown not to be on the "main line." Thus if one player P has a move leading to a position of value v, any alternative or future move that would let the opponent produce a value v' worse for P than v need not be considered, since P can (and should) always make choices in a way that avoid the value v'.

The assumption in the literature has been that terms such as "better" and "worse" refer to comparisons made using a total order; there has been almost no consideration of games where payoffs may be incomparable. As an example, imagine a game involving a card selected at random from a standard 52-card deck. If I make move m_1, I will win the game if the card is an ace. If I make

This work has been supported by DARPA/Rome Labs under contracts F30602-95-1-0023 and F30602-97-1-0294.

move m_2, I will win the game if the card is a spade. Since there are more spades in the deck than aces, m_2 is presumably the better move.

The conventional way to analyze this situation is to convert both conditions to probabilities, saying that m_1 has a payoff of $1/13$ and m_2 a payoff of $1/4$. The second payoff is better, so I make move m_2.

Now imagine, however, that I am playing a more complicated game. This game involves two subgames; in order to win the overall game, I need to win both subgames. The first subgame is as described in the previous paragraph. For the second subgame, m_1 wins if the card is an ace as before; m_2 wins if the card is a heart. It is clear that m_2 has no chance at all of winning the overall game (a card cannot be a heart and a spade both), so that m_1 is now to be preferred.

This example should make it clear that when we assign a number to a move such as m_1 or m_2, the assignment is only truly meaningful at the root of the game tree. At internal nodes, this game requires that we understand the context in which a particular move wins or loses, as opposed to simply the probability of that context. The contexts in which different moves win will not always be comparable, although they can be forced to be comparable by converting each context into a real number probabilistically. Doing so makes sense at the root of the game tree, but not at internal nodes or other situations where the contexts must be combined in some fashion.

Our goal in this paper is to analyze games while working with the most natural labels available. These may be real numbers, tuples of real numbers (natural if the payoff function involves multiple attributes), contexts (often natural in games of incomplete information), or other values. Given a game described in these terms, under what conditions can we continue to use α-β pruning to reduce the search space?

We will answer this question in the next three sections. Section 2 begins by introducing general operations on the value set that are needed if the notion of a game tree itself is to be meaningful. In Section 3, we go on to show that the validity of shallow α-β pruning is equivalent to the condition that the payoff values are taken from a mathematical structure known as a *lattice*. In Section 4, we show further that deep α-β pruning is valid if and only if the lattice in question is distributive. Section 5 describes an application of our techniques to card play problems arising in contract bridge. Related work is discussed briefly in Section 6 and concluding remarks are contained in Section 7.

2. Structure and Definitions

By a *game*, we will mean basically a set of rules that determine play from a given initial position I. We will assume that all games are two-player, involving a minimizer (often denoted 0) and a maximizer (often denoted 1). The "rules" of the game consist of a successor function s that takes a position p and returns the set $s(p)$ of possible subsequent positions. If $s(p) = \varnothing$, p is a terminal position and is assigned a value by an evaluation function e.

Definition 2.1. A *game* is an octuple

$$(P, V, I, w, s, e, f_+, f_-)$$

such that:

(i) P is the set of possible positions in the game.
(ii) V is the set of values for the game.
(iii) $I \in P$ is the initial position of the game.
(iv) $w : P \to \{0, 1\}$ is a function indicating which player is to move in each position in P.
(v) $s : P \to \mathcal{P}(P)$ gives the successors of each position in P.
(vi) $e : s^{-1}(\varnothing) \to V$ evaluates each terminal position in P.
(vii) $f_+ : \mathcal{P}(V) \to V$ and $f_- : \mathcal{P}(V) \to V$ are the combination functions for the maximizer and minimizer respectively.

Most games that have been discussed in the AI literature take V to be the set $\{-1, 0, +1\}$, with -1 being a win for the minimizer, $+1$ a win for the maximizer, and 0 a draw. The functions f_+ and f_- conventionally return the maximum and minimum of their set-valued arguments.

Note, incidentally, that the assumption that f_+ and f_- have sets as arguments is not without content. It says, for example, that there is no advantage to either player to have multiple winning options in a given position; one winning option suffices.

A game is finite if there is no infinite sequence of legal moves starting at the initial position. Formally:

Definition 2.2. A game $(P, V, I, w, s, e, f_+, f_-)$ will be called *finite* if there is some integer N such that there is no sequence $p_0, \ldots, p_N$ with $p_0 = I$ and $p_i \in s(p_{i-1})$ for $i = 1, \ldots, N$.

Given the above definitions, we can use the minimax algorithm to assign values to nonterminal nodes of the game tree:

Definition 2.3. Let $G = (P, V, I, w, s, e, f_+, f_-)$ be a finite game. By the *evaluation function for G* we will mean that function $\overline{e}$ defined recursively by

$$\overline{e}(p) = \begin{cases} e(p), & \text{if } s(p) = \varnothing; \\ f_+\{\overline{e}(p') | p' \in s(p)\} & \text{if } w(p) = 1; \\ f_-\{\overline{e}(p') | p' \in s(p)\} & \text{if } w(p) = 0. \end{cases} \qquad (2\text{–}1)$$

The *value* of G will be defined to be $\overline{e}(I)$.

Proposition 2.4. *The evaluation function $\overline{e}$ is well defined for finite games.*

Proof. The proof proceeds by induction on the distance of a given position p from the fringe of the game tree. ☐

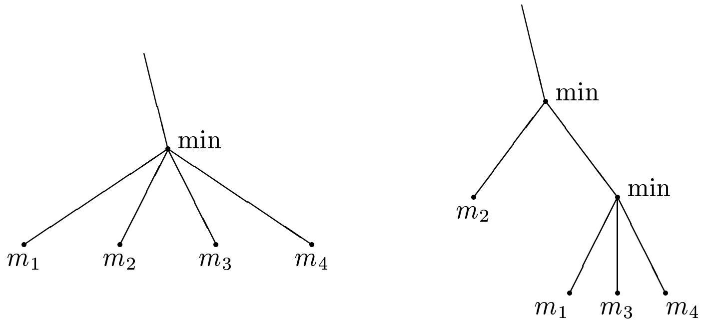

Figure 1. Equivalent games?

The expression (2–1) is, of course, just the usual minimax algorithm, reexpressed as a definition. In keeping with our overall goals, we have replaced the usual max and min operations with f_+ and f_-.

Consider now the two game trees in Figure 1, where none of the m_i are intended to be necessarily terminal. Are these two games always equivalent?

We would argue that they are. In the game on the left, the minimizer needs to select among the four options m_1, m_2, m_3, m_4. In the game on the right, he needs to first select whether or not to play m_2; if he decides not to, he must select among the remaining options. Since the minimizer has the same possibilities in both cases, we assume that the values assigned to the games are the same.

From a more formal point of view, the value of the game on the left is $f_-(m_1, m_2, m_3, m_4)$ and that of the game on the right is $f_-(m_2, f_-(m_1, m_3, m_4))$, where we have abused notation somewhat, writing m_i for the value of the node m_i as well.

Definition 2.5. A game will be called *simple* if for any $x \in v \subseteq V$,

$$f_+\{x\} = f_-\{x\} = x$$

and also

$$f_+(v) = f_+\{x, f_+(v - x)\} \quad \text{and} \quad f_-(v) = f_-\{x, f_-(v - x)\}.$$

We have augmented the condition developed in the discussion of Figure 1 with the assumption that if a player's move in a position p is forced (so that p has a unique successor), then the value before and after the forced move is the same.

Proposition 2.6. *For any simple game, there are binary functions* $\wedge$ *and* $\vee$ *from* V *to itself that are commutative, associative and idempotent*[1] *and such that*

$$f_+\{v_0, \ldots, v_m\} = v_0 \vee \cdots \vee v_m \quad \textit{and} \quad f_-\{v_0, \ldots, v_m\} = v_0 \wedge \cdots \wedge v_m.$$

Proof. Induction on m. □

[1]A binary function f is called *idempotent* if $f(a, a) = a$ for all a.

When referring to a simple game, we will typically replace the functions f_+ and f_- by the equivalent binary functions $\vee$ and $\wedge$. We assume throughout the rest of this paper that all games are simple and finite.[2]

The binary functions $\vee$ and $\wedge$ now induce a partial order $\leq$, where we will say that $x \leq y$ if and only if $x \vee y = y$. It is not hard to see that this partial order is reflexive ($x \leq x$), antisymmetric ($x \leq y$ and $y \leq x$ if and only if $x = y$) and transitive. The operators $\vee$ and $\wedge$ behave like greatest lower bound and least upper bound operators with regard to the partial order.

We also have the following:

Proposition 2.7. *Whenever* $S \subseteq T$, $f_+(S) \leq f_+(T)$ *and* $f_-(S) \geq f_-(T)$. $\square$

In other words, assuming that the minimizer is trying to reach a low value in the partial order and the maximizer is trying to reach a high one, having more options is always good.

3. Shallow Pruning

What about α-β pruning in all of this? Let us begin with shallow pruning, shown in Figure 2.

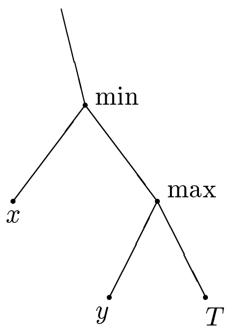

Figure 2. T can be pruned (shallowly) if $x \leq y$.

The idea here is that if the minimizer prefers x to y, he will never allow the maximizer even the possibility of selecting between y and the value of the subtree rooted at T. After all, the value of the maximizing node in the figure is $y \vee \overline{e}(T) \geq y \geq x$, and the minimizer will therefore always prefer x.

We will not give a precise definition of α-β pruning here, because the formal definition is fairly intricate if the game tree can be a graph (and we have nowhere excluded that). Instead, we observe simply that:

[2]We also assume that the games are sufficiently complex that we can find in the game tree a node with any desired functional value, e.g., $a \wedge (b \vee c)$ for specific a, b and c. Were this not the case, none of our results would follow. As an example, a game in which the initial position is also terminal surely admits pruning of all kinds (since the game tree is empty) but need not satisfy the conclusions of the results in the next sections.

Definition 3.1 (Shallow α-β pruning)**.** A game G will be said to *allow shallow α-β pruning for the minimizer* if

$$x \wedge (y \vee T) = x \tag{3–1}$$

for all $x, y, T \in V$ with $x \leq y$. The game will be said to *allow shallow α-β pruning for the maximizer* if

$$x \vee (y \wedge T) = x \tag{3–2}$$

for all $x, y, T \in V$ with $x \geq y$. We will say that G *allows shallow pruning* if it allows shallow α-β pruning for both players.

As we will see shortly, the expressions (3–1) and (3–2) describing shallow pruning are identical to what are more typically known as *absorption identities.*

Definition 3.2. Suppose V is a set and $\wedge$ and $\vee$ are two binary operators on V. The triple $(V, \wedge, \vee)$ is called a *lattice* if $\wedge$ and $\vee$ are idempotent, commutative and associative, and satisfy the *absorption identities* in that for any $x, y \in V$,

$$x \vee (x \wedge y) = x, \tag{3–3}$$

$$x \wedge (x \vee y) = x. \tag{3–4}$$

Definition 3.3. A lattice $(V, \wedge, \vee)$ is called *distributive* if $\wedge$ and $\vee$ distribute with respect to one another, so that

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z), \tag{3–5}$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z). \tag{3–6}$$

Lemma 3.4. *Each of* (3–3) *and* (3–4) *implies the other. Each of* (3–5) *and* (3–6) *implies the other.*

These are well known results from lattice theory [Grätzer 1978].

Proposition 3.5. *For a game G, the following conditions are equivalent:*

(i) *G allows shallow α-β pruning for the minimizer.*
(ii) *G allows shallow α-β pruning for the maximizer.*
(iii) *G allows shallow pruning.*
(iv) *$(V, \wedge, \vee)$ is a lattice.*

Proof. We show that the first and fourth conditions are equivalent; everything else follows easily.

If G allows shallow α-β pruning for the minimizer, we take $x = a$ and $y = T = a \vee b$ in (3–1). Clearly $x \leq y$ so we get

$$a \wedge (y \vee y) = a \wedge y = a \wedge (a \vee b) = a$$

as in (3–4).

For the converse, if $x \leq y$, then $x \wedge y = x$ and

$$x \wedge (y \vee T) = (x \wedge y) \wedge (y \vee T) = x \wedge (y \wedge (y \vee T)) = x \wedge y = x. \qquad \square$$

4. Deep Pruning

Deep pruning is a bit more subtle. An example appears in Figure 3.

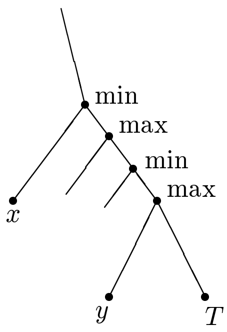

Figure 3. T can be pruned (deeply) if $x \leq y$.

As before, assume $x \leq y$. The argument is as described in the introduction: Given that the minimizer has a guaranteed value of x at the upper minimizing node, there is no way that a choice allowing the maximizer to reach y can be on the main line; if it were, then the maximizer could get a value of at least y.

Definition 4.1 (Deep α-β pruning)**.** A game G will be said to *allow α-β pruning for the minimizer* if for any $x, y, T, z_1, \ldots, z_{2i} \in V$ with $x \leq y$,

$$x \wedge (z_1 \vee (z_2 \wedge \cdots \vee (z_{2i} \wedge (y \vee T))) \cdots) = x \wedge (z_1 \vee (z_2 \wedge \cdots \vee z_{2i}) \cdots).$$

The game will be said to *allow α-β pruning for the maximizer* if

$$x \vee (z_1 \wedge (z_2 \vee \cdots \wedge (z_{2i} \vee (y \wedge T))) \cdots) = x \vee (z_1 \wedge (z_2 \vee \cdots \wedge z_{2i}) \cdots).$$

We will say that G *allows pruning* if it allows α-β pruning for both players.

As before, the prune allows us to remove the dominated node (y in Figure 3) and all of its siblings.

Note that the fact that a game allows shallow α-β pruning does not mean that it allows pruning in general, as the following counterexample shows.

This game is rather like the game of the introduction, except only the suit of the card matters. In addition, the maximizer (but only the maximizer) is allowed to specify that the card must be one of two suits of his choosing.

The values we will assign the game are built from the primitives, "Win (for the maximizer) if the card is a club," "Win if the card is a diamond," and so on. The conjunction of two such values is a loss for the maximizer, since the card cannot be of two suits. The disjunction of two such values is a win for the maximizer, since he can require that the card be one of the two suits in question. The lattice of values is thus the one appearing in Figure 4. The maximizing function $\vee$ moves up the figure; the minimizing function $\wedge$ moves down.

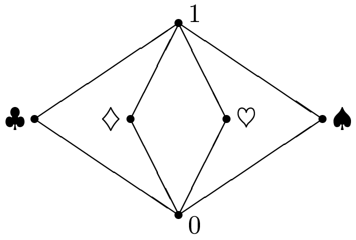

Figure 4. Values in a game where deep pruning fails.

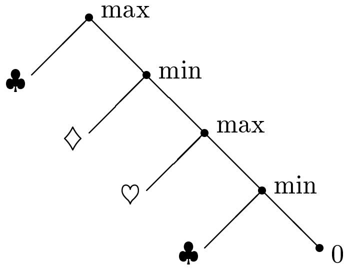

Figure 5. The deep pruning counterexample.

Consider now the game tree shown in Figure 5. If we evaluate it as shown, the backed up values are (from the lower right) 0, $\heartsuit$, 0, and $\clubsuit$, so that the maximizer wins if and only if the card is a club. But if we use deep α-β pruning to remove the 0 in the lower right (since its sibling has the same value $\clubsuit$ as the value in the upper left), we get values of $\clubsuit$, 1, $\diamondsuit$ and 1, so that the maximizer wins outright.

The problem is that we can't "push" the value $\clubsuit \wedge 0$ past the $\heartsuit$ to get to the $\clubsuit$ near the root. Somewhat more precisely, the problem is that

$$\heartsuit \vee (\clubsuit \wedge 0) \neq (\heartsuit \wedge \clubsuit) \vee (\heartsuit \wedge 0).$$

This suggests the following:

Proposition 4.2. *For a game G, the following conditions are equivalent:*

(i) *G allows α-β pruning for the minimizer.*
(ii) *G allows α-β pruning for the maximizer.*
(iii) *G allows pruning.*
(iv) *$(V, \wedge, \vee)$ is a distributive lattice.*

Proof. As before, we show only that the first and fourth conditions are equivalent. Since pruning implies shallow pruning (take $i = 0$ in the definition), it follows that the first condition implies that $(V, \wedge, \vee)$ is a lattice.

From deep pruning for the minimizer with $i = 1$, we have that if $x \leq y$, then for any z_1, z_2, T,

$$x \wedge (z_1 \vee (z_2 \wedge (y \vee T))) = x \wedge (z_1 \vee z_2)$$

Now take $y = T = x$ to get

$$x \wedge (z_1 \vee (z_2 \wedge x)) = x \wedge (z_1 \vee z_2). \tag{4–1}$$

It follows that each top level term in the left hand side of (4–1) is greater than or equal to the right hand side; specifically

$$z_1 \vee (z_2 \wedge x) \geq x \wedge (z_1 \vee z_2). \tag{4–2}$$

We claim that this implies that the lattice in question is distributive.

To see this, let $u, v, w \in V$. Now take $z_1 = u \wedge w$, $z_2 = v$ and $x = w$ in (4–2) to get

$$(u \wedge w) \vee (v \wedge w) \geq w \wedge ((u \wedge w) \vee v). \tag{4–3}$$

But $v \vee (u \wedge w) \geq w \wedge (v \vee u)$ is an instance of (4–2), and combining this with (4–3) gives us

$$\begin{aligned} (u \wedge w) \vee (v \wedge w) &\geq w \wedge ((u \wedge w) \vee v) \\ &\geq w \wedge w \wedge (v \vee u) \\ &= w \wedge (v \vee u). \end{aligned}$$

This is the hard direction; $w \wedge (v \vee u) \geq (u \wedge w) \vee (v \wedge w)$ for any lattice because $w \wedge (v \vee u) \geq u \wedge w$ and $w \wedge (v \vee u) \geq v \wedge w$ individually. Thus $w \wedge (v \vee u) = (u \wedge w) \vee (v \wedge w)$, and deep pruning implies that the lattice is distributive.

For the converse, if the lattice is distributive and $x \leq y$, then

$$\begin{aligned} x \wedge (z_1 \vee (z_2 \wedge (y \vee T))) &= (x \wedge z_1) \vee (x \wedge z_2 \wedge (y \vee T)) \\ &= (x \wedge z_1) \vee (x \wedge z_2) \\ &= x \wedge (z_1 \vee z_2), \end{aligned}$$

where the second equality is a consequence of the fact that $x \leq (y \vee T)$, so that $x = x \wedge (y \vee T)$. This validates pruning for $i = 1$; deeper cases are similar. □

5. Bridge

To test our ideas in practice, we built a card-playing program for the game of contract bridge. Previous authors [Ginsberg 1996a; Ginsberg 1996b; Ginsberg 1999; Levy 1989] have too often considered only the perfect-information variant of this game; the problems with this approach have been pointed out by others [Frank and Basin 1998]. The proposed solution to this problem has been to search in the space of possible plans for playing a given deal or position, but the complexity of finding an optimal plan is NP-complete in the size (not depth) of the game tree, which is prohibitive. As a result, Frank et.al. report that it takes an average of 571 seconds to run an approximate search algorithm on problems of size 13 (running on a 300 MHz UltraSparc II) [Frank *et al.* 1998].

We built a search engine that is capable of solving problems such as these exactly. The value assigned to a fringe node is not simply the number of tricks that can be taken by one side or the other on a particular bridge hand, but a combination of the number of tricks and information that has been acquired about the hidden hands (e.g., West has the ♡7 but not the ♠Q, and I can take seven tricks).

The defenders are assumed to be minimizing and to be playing with perfect information, so that their combination function is simple logical disjunction. As an example, if the nondefending side can take four tricks (and hence one, two or three as well) in one case, and only three tricks in another, the greatest lower bound of the two values is that the nondefending side can take three tricks – the disjunction.

The declaring side is assumed to be maximizing and is playing with incomplete information, so that they cannot simply take the best of two potentially competing outcomes. Instead, the maximizer works with values of the form $\texttt{choose}(v_1, \ldots, v_n)$ where the v_i are the values assigned to the lines from which a choice must be made.

When these two operators are used to construct a (distributive) lattice in the obvious way, the ideas we have discussed can be applied to solve bridge problems under the standard assumption [Frank and Basin 1998] that the maximizer is playing with incomplete information while the minimizer is playing perfectly. The performance of the resulting algorithm is vastly improved over that reported by Frank. Endings with up to 32 cards are solved routinely in a matter of a few seconds (Frank et.al. could only deal with 13 cards), and endings with 13 cards are solved essentially instantly on a 500-MHz Pentium III. In addition, the results computed are exact as opposed to approximate.

6. Related Work

There appear to be two authors who have discussed issues related to those presented here.

Dasgupta et. al. examines games where values are taken from $\mathcal{R}^n$ instead of simply $\mathcal{R}$ [Dasgupta *et al.* 1996]. Since any distributive lattice can be embedded in $\mathcal{R}^n$ for some n, it follows from Dasgupta's results that if the values are taken from a distributive lattice, then α-β pruning can be applied. The converse, along with our results regarding nondistributive lattices, are not covered by this earlier work.

Müller considers games with values taken from arbitrary partially ordered sets, evaluating such games by taking a threshold value and then converting the original game to a 0-1 game by calling a position a win if its value exceeds the threshold and a loss otherwise [Müller 2000].

The problem with this approach is that you need some way to produce the threshold value; Müller suggests mapping the partial order in question into a discrete total order (the integers), and then using the values in that total order to produce thresholds for the original game.

While this appears to work in some instances (Müller examines capturing races in Go), it is unlikely to work in others. In bridge, for example, the natural mapping is from a context in which a hand can be made to the (discretized) probability that the context actually occurs. Unfortunately, it is possible to have three contexts c_1, c_2 and c_3 where the probability of c_1 or c_2 in isolation is less than the probability of c_3, but the probability of c_1 and c_2 together is greater than the probability of c_3. This suggests that there is no suitable linearization of the partial order discussed in the previous section, so that Müller's ideas will be difficult to apply.

7. Conclusion

If adversary search is to be effective in a setting where positional values have more structure than simple real numbers, we need to understand conditions under which proven AI algorithms can still be applied. For α-β pruning, this paper has characterized those conditions exactly: Shallow pruning remains valid if and only if the values are taken from a lattice, while deep pruning is valid if and only if the lattice in question is distributive. We also showed the somewhat surprising result that pruning is valid for one player if and only if it is valid for both.

We used the theoretical work described here to implement a new form of cardplay engine for the game of contract bridge. This implementation uses the standard bridge assumption that the defenders will play perfectly while the declarer may not, and avoids the Monte Carlo techniques used by previous authors. Compared to previous programs with this goal, we are able to compute exact results in a small fraction of the time previously needed to compute approximate ones.

References

[Dasgupta *et al.* 1996] Pallab Dasgupta, P. P. Chakrabarti, and S. C. DeSarkar. Searching game trees under a partial order. *Artificial Intelligence*, 82:237–257, 1996.

[Frank and Basin 1998] Ian Frank and David Basin. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100:87–123, 1998.

[Frank *et al.* 1998] Ian Frank, David Basin, and Hitoshi Matsubara. Finding optimal strategies for imperfect information games. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 500–507, 1998.

[Ginsberg 1996a] Matthew L. Ginsberg. How computers will play bridge. *The Bridge World*, 1996.

[Ginsberg 1996b] Matthew L. Ginsberg. Partition search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[Ginsberg 1999] Matthew L. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.

[Grätzer 1978] George Grätzer. *General Lattice Theory.* Birkhäuser Verlag, Basel, 1978.

[Levy 1989] David N. L. Levy. The million pound bridge program. In D. N. L. Levy and D. F. Beal, editors, *Heuristic Programming in Artificial Intelligence*, Asilomar, CA, 1989. Ellis Horwood.

[Müller 2000] M. Müller. Partial order bounding: A new approach to evaluation in game tree search. Technical Report TR-00-10, Electrotechnical Laboratory, Tsukuba, Japan, 2000. Available from http://web.cs.ualberta.ca/~mmueller/publications.html.

Matthew L. Ginsberg
CIRL
1269 University of Oregon
Eugene, OR 97403-1269
United States
ginsberg@cirl.uoregon.edu

Alan Jaffray
jaffray@pobox.com