Artificial Intelligence 48 (1991) 99–111
Elsevier

Research Note

Multi-player alpha-beta pruning

Richard E. Korf*

Computer Science Department, University of California, Los Angeles, Los Angeles, CA 90024, USA

Received August 1989
Revised June 1990

Abstract

Korf, R.E., Multi-player alpha-beta pruning (Research Note), Artificial Intelligence 48 (1991) 99–111.

We consider the generalization of minimax search with alpha-beta pruning to non-cooperative, perfect-information games with more than two players. The minimax algorithm was generalized in [2] to the maxn algorithm applied to vectors of n-tuples representing the evaluations for each of the players. If we assume an upper bound on the sum of the evaluations for each player, and a lower bound on each individual evaluation, then shallow alpha-beta pruning is possible, but not deep pruning. In the best case, the asymptotic branching factor is reduced to $(1+\sqrt{4b-3})/2$. In the average case, however, pruning does not reduce the asymptotic branching factor. Thus, alpha-beta pruning is found to be effective only in the special case of two-player games. In addition, we show that it is an optimal directional algorithm for two players.

1. Introduction

Minimax search with alpha-beta pruning is the predominant algorithm employed by two-player game programs [1, 3, 5]. Figure 1 shows a game tree, where squares represent maximizing nodes and circles correspond to minimizing nodes, along with its minimax value, bounds on interior nodes, and those branches pruned by alpha-beta.

In this paper, we consider the generalization of alpha-beta pruning to non-cooperative, perfect-information games with more than two players. For example, Chinese Checkers can involve up to six different players moving

* Section 3 of this paper represents a more complete treatment of Section 3 of a paper by the same author entitled, "Generalized game trees", that appeared in the *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, MI (1989).

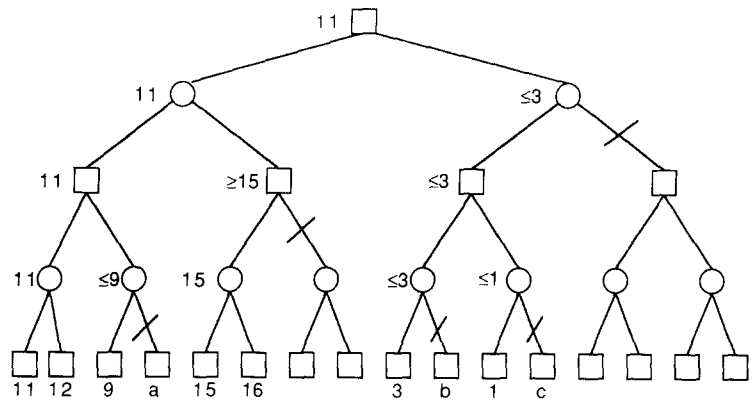

Fig. 1. Two-player alpha-beta pruning.

alternately. As another example, Othello can easily be extended to an arbitrary number of players by having different colored pieces for each player, and modifying the rules such that whenever a mixed row of opposing pieces is flanked on both sides by two pieces of the same player, then all the pieces are captured by the flanking player.

2. Maxn algorithm

Luckhardt and Irani [2] extended minimax to multi-player games, calling the resulting algorithm max^n. For typographical convenience we refer to it as *maxn*. They assume that the players alternate moves, that each player tries to maximize his or her perceived return, and is indifferent to the returns of the remaining players. At the frontier nodes, an evaluation function is applied that returns an N-tuple of values, where N is the number of players, with each component corresponding to the estimated merit of the position with respect to one of the players. Then, the value of each interior node where player i is to move is the entire N-tuple of the child for which the ith component is a maximum. Figure 2 shows a maxn tree for three players, with the corresponding maxn values.

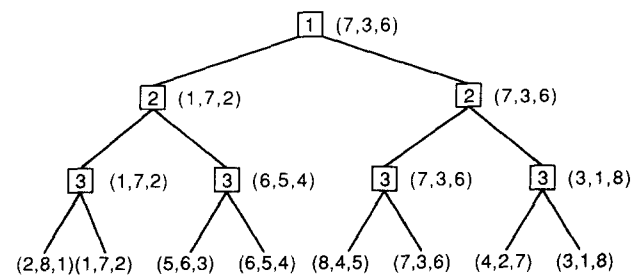

Fig. 2. Three-player maxn game tree.

More formally, define $M(x)$ to be the static heuristic value of node x, and $M(x, p)$ to be the backed-up maxn value of node x, given that player p is to move at node x. $M_i(x, p)$ is the component of $M(x, p)$ that corresponds to the return for player i. We can then define the maxn value of a node recursively as follows:

$$M(x, p) = \begin{cases} M(x), & \text{if } x \text{ is a frontier node,} \\ M(x_i, p'), & \text{otherwise,} \end{cases}$$

such that $M_p(x_i, p') = \max M_p(x_i, p')$ where x_i are the children of node x, p' is the player that follows player p in the move order, and ties are broken in favor of the leftmost node.

For example, an evaluation function for multi-player Othello might return the number of pieces for each player on the board at any given point.

Minimax can be viewed as a special case of maxn for two players. The evaluation function returns an ordered pair of x and $-x$, and each player maximizes its component at its moves.

Luckhardt and Irani [2] observed that at nodes where player i is to move, only the ith component of the children need be evaluated. At best, this can produce a constant factor speedup, but it may be no less expensive to compute all components than to compute only one. They correctly concluded that without further assumptions on the values of the components, pruning of entire branches is not possible with more than two players. Thus, they did not explore such pruning any further.

They used the terms "shallow pruning" and "deep pruning" to refer to their techniques of avoiding some partial evaluations. Since these terms had previously been used to describe actual tree pruning in the alpha-beta literature [1], we will use the original meanings of both these terms, at the cost of inconsistency with Luckhardt's and Irani's terminology [2].

3. Alpha-beta in multi-player games

If there is an upper bound on the sum of all components of a tuple, and there is a lower bound on the values of each component, then actual tree pruning is possible. The first condition is a weaker form of the standard constant-sum assumption, which is in fact required for two-player alpha-beta pruning. The second is equivalent to assuming a lower bound of zero on each component, since any other lower bound can be shifted to zero by subtracting it from every component. Most practical evaluation functions will satisfy both these conditions, since violating them implies that the value of an individual component can be unbounded in at least one direction. For example, in the evaluation function described above for multi-player Othello, no player can have less than zero pieces on the board, and the total number of pieces on the

board is the same for all nodes at the same level in the game tree, since exactly one piece is added at each move.

3.1. Immediate pruning

The simplest kind of pruning possible under these assumptions occurs when player i is to move, and the ith component of one of its children equals the upper bound on the sum of all components. In that case, all remaining children can be pruned, since no child's ith component can exceed the upper bound on the sum. We will refer to this as *immediate pruning*. This is equivalent to situations in the two-player case when a child of a Max node has a value of ∞, or a child of a Min node has a value of $-\infty$.

3.2. Shallow pruning

A more complex situation is called *shallow pruning* in the alpha-beta literature. Figure 3 shows an example of shallow pruning in a three-player game, where the upper bound on the sum of each component is 9. Note that in this particular example, the sum of each component is exactly 9, but an upper bound is all that is required. Evaluating node b results in a lower bound of 3 on the first component of node a, since player one is to move at node a. This implies an upper bound on each of the remaining components of $9-3=6$. Evaluating node g produces a lower bound of 7 on the second component of node f, since player two is to move at node f. Similarly, this implies an upper bound on the remaining components of $9-7=2$. Since the upper bound (2) on the first component of node f is less than or equal to the lower bound on the first component of node a (3), player one won't choose node f and its remaining children can be pruned. Similarly, evaluating node i causes its remaining brothers to be pruned. This is similar to the pruning in the left subtree of Fig. 1.

The procedure *Shallow* takes a *Node* to be evaluated, the *Player* to move at that node, and an upper *Bound* on the component of the player to move, and returns an N-tuple that is the maxn value of the node. *Sum* is the global upper bound on the sum of all components of an N-tuple, and all components are assumed to be non-negative. Initially, *Shallow* is called with the root of the

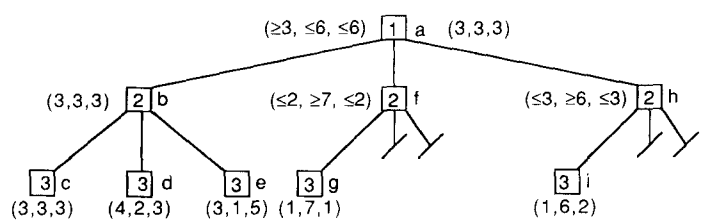

Fig. 3. Shallow pruning in three-player game tree.

tree, the player to move at the root, and *Sum*. Note that the shallow pruning procedure includes immediate pruning as a special case.

```
Shallow(Node, Player, Bound)
IF Node is terminal, RETURN static value
Best = Shallow(first Child, next Player, Sum)
FOR each remaining Child
  IF Best[Player] > = Bound, RETURN Best
  Current = Shallow(Child, next Player, Sum - Best[Player])
  IF Current[Player] > Best[Player], Best = Current
RETURN Best
```

3.3. Correctness of shallow pruning procedure

Here we establish the correctness of the shallow pruning procedure, in the sense that it computes the maxn value defined in Section 2.

Theorem 1. *If the evaluation of any position for any player is non-negative, and the sum of all player's evaluations in any given position is less than or equal to sum, and ties are broken in favor of the leftmost node, then Shallow*$(a, p, sum) = M(a, p)$*, for any node a and player p.*

Proof. Since the only difference between the maxn procedure and the shallow pruning procedure is that it doesn't examine certain nodes, it suffices to show that the pruned nodes can have no effect on the maxn value of the root. There are two types of pruning implemented in the shallow pruning procedure, immediate pruning and shallow pruning.

Figure 4 shows the generic case of immediate pruning. Without loss of generality, we assume that player one is to move at the root. In order for immediate pruning to occur, $M_1(b, 2) = sum$, for some child b of node a. Since all components are assumed to be non-negative, *sum* is the maximum possible value of any individual component, and the remaining components must be zero. Therefore $sum = M_1(b, 2) = \max M_1(a_i, 2)$, where a_i is a child of node a. Thus, $M(a, 1) = M(b, 2)$, and the remaining children of node a need not be examined.

Figure 5 shows the generic case of shallow pruning. Again we assume without loss of generality that player one is to move at the root, followed by

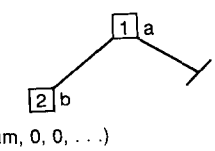

Fig. 4. Generic case of immediate pruning.

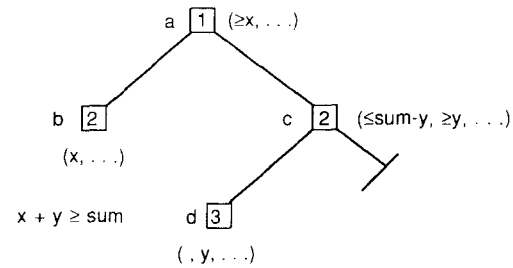

Fig. 5. Generic case of shallow pruning.

players two and three in turn. $M_1(b, 2) = x$, $M_2(d, 3) = y$, and $x + y \geq sum$, in order for shallow pruning to occur. Since player two is to move at node c, $M_2(c, 2) = \max M_2(c_i, 3)$, where c_i is a child of node c. Therefore, $M_2(c, 2) \geq M_2(d, 3) = y$. Since the sum of all components cannot exceed *sum*, and all components are non-negative, $M_1(c, 2) \leq sum - y \leq x$, since $x + y \geq sum$. Since $M_1(b, 2) = x \geq M_1(c, 2)$, and ties are broken in favor of leftmost nodes, $M(a, 1) \neq M(c, 2)$, and node c can have no effect on the maxn value of node a. Therefore, the remaining children of node c can be pruned.

Since neither immediate nor shallow pruning eliminate nodes that can effect the maxn value of the root node, and the shallow pruning procedure returns the maxn value of the nodes it has examined, it correctly calculates the maxn value of the root node. □

3.4. *Failure of deep pruning*

In a two-player game, alpha-beta pruning allows an additional type of pruning known as *deep pruning*. For example, in Fig. 1, nodes b and c are pruned based on bounds inherited from their great-great-grandparent, the root in this case. In general, deep pruning refers to pruning a node based on a bound inherited from its great-grandparent, or any more distant ancestor. In a two-player game tree, it can only occur in trees of height four or greater. Surprisingly, deep pruning does not generalize to more than two players.

Figure 6 illustrates the problem. Again, the upper bound on the sum of each component is 9. Evaluating node b produces a lower bound of 5 on the first component of node a and hence an upper bound of $9 - 5 = 4$ on the remaining components. Evaluating node e results in a lower bound of 5 on the third component of node d and hence an upper bound of $9 - 5 = 4$ on the remaining components. Since the upper bound of 4 on the first component of node d is less than the lower bound of 5 on the first component of node a, the value of node d cannot become the value of node a. Thus, we are tempted to prune node f.

With three players, however, the value of node f could effect the value of the root, depending on the value of node g. For example, if the value of node f

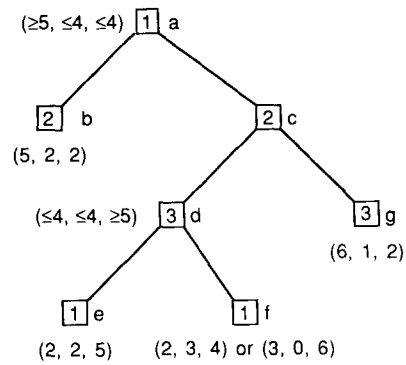

Fig. 6. Failure of deep pruning for three players.

were $(2, 3, 4)$, the value of node d would be $(2, 2, 5)$, the value of node c would be $(2, 2, 5)$, and the value of node a would be $(5, 2, 2)$. On the other hand, if the value of node f were $(3, 0, 6)$, then the value of node d would be $(3, 0, 6)$, the value of node c would be $(6, 1, 2)$, and the value of node a would be $(6, 1, 2)$. Thus, even though the value of node f cannot *be* the maxn value of the root, it can *affect* it. Hence, it cannot be pruned.

3.5. *Optimality of shallow pruning*

Given the failure of deep pruning in this example, is there a more restricted form of pruning that is valid, or is shallow pruning the best we can do? The answer is the latter, as expressed by the following theorem:

Theorem 2. *Every directional algorithm that computes the maxn value of a game tree with more than two players must evaluate every terminal node evaluated by shallow pruning under the same ordering.*

A directional algorithm [4] is one in which the order of node evaluation is independent of the value of the nodes, and once a node is pruned it can never be revisited. For example, a strictly left-to-right algorithm is directional.

Proof sketch. Since the actual proof given below is somewhat tedious and not very revealing, we first present here an overview and example of the argument. The main idea is illustrated by the construction in Fig. 7, which shows a 3-player, 6-level tree. We assume that node n is evaluated by shallow pruning but pruned by another algorithm. We then show that the value of every node above it depends on the value of node n. The letters to the left of the path from the root to node n represent the greatest lower bounds on the components corresponding to the player to move at each node. Since by assumption node n is evaluated by shallow pruning, it must be the case that for any two

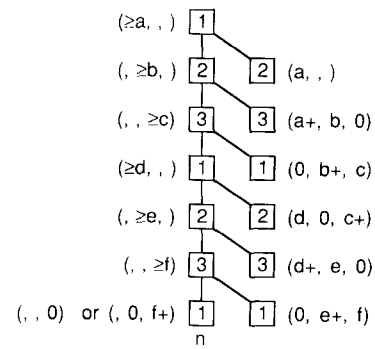

Fig. 7. Proof sketch of optimality of shallow pruning.

consecutive bounds x and y, $x + y$ is strictly less than the global upper bound. Since the decision to skip node n is made before any of the nodes to the right of the path are examined, we are free to choose any values for these nodes consistent with the global bounds. For this purpose, $x+$ represents a value greater than x by an arbitrarily small amount. The reader is encouraged to assign each of the two alternative values to node n, and then compute the maxn value of the root, to see that it is different in the two cases. The propagation of values up the tree can be viewed as a "zipper" effect in the sense that the original order of the "teeth" (nodes) at the bottom determines the order of the teeth at the top, even though no individual tooth can move very far. The formal proof below is by induction on the height of the tree and generalizes the result to an arbitrary number of players greater than two.

Proof (see Fig. 8). Assume the converse. Namely, that there exists a directional maxn algorithm A, and a minimax tree T with N players, with a leaf node X^0 that is evaluated by shallow pruning but not by algorithm A. Since both shallow pruning and algorithm A are directional, and by assumption visit nodes in the same order, (re)order the nodes in T from left to right in the order they are visited, without loss of generality. Let X^i refer to the ancestor of node X^0 at height i in the tree, on the unique path from the root to X^0, and also to its maxn value, depending on the context. Let X^i_j refer to the jth component of the maxn value of node X^i. Thus, $X^i = (X^i_0, \ldots, X^i_j, \ldots, X^i_n)$. Let $p(i)$ be the player to move at level i in the tree. Thus, $X^i_{p(i)}$ is the maximum value of $W^{i-1}_{p(i)}$ among all the children W^{i-1} of X^i.

Consider the state of the shallow pruning algorithm just before it evaluates node X^0. It consists of the path from the root to node X^0, plus a set of bounds α_i, where α_i is equal to the maximum value of $W^{i-1}_{p(i)}$ among all the children W^{i-1} of X^i to the left of X^{i-1}. In other words, α_i is the greatest lower bound so far on $X^i_{p(i)}$. If at any level i, X^{i-1} has no brothers to its left, then α_i will simply equal the global lower bound on each component. Without loss of generality,

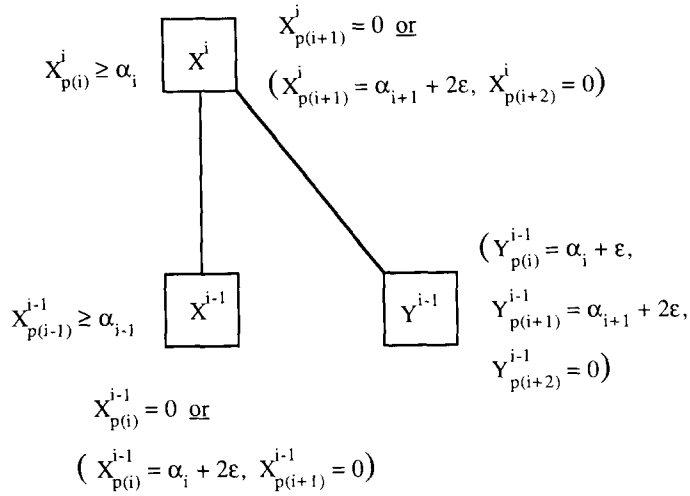

Fig. 8. Inductive proof of optimality of shallow pruning.

let the global lower bound on each component be zero and the global upper bound on the sum of all components be one. Any other bounds can always be mapped to this range without affecting either algorithm since only relative values matter.

The assumption that X^0 is evaluated by shallow pruning constrains the values of the α_i. First, each of the α_i must be strictly less than one. Otherwise, the remaining children of X^i would be immediately pruned, pruning X^0 as well. Furthermore, each lower bound α_i induces a corresponding upper bound of $1 - \alpha_i$ on the remaining components of X^i. If this upper bound on $X^i_{p(i-1)}$ is less than or equal to the lower bound of α_{i-1} on $X^{i-1}_{p(i-1)}$, then the remaining children of X^i would be pruned by shallow pruning, pruning X^0 as well. Thus, for all $i \geq 1$, $1 - \alpha_i > \alpha_{i-1}$, or $\alpha_i + \alpha_{i-1} < 1$.

Let Y^i be a brother of X^i to its right. When either algorithm is about to consider X^0, the values and even the existence of the Y^i nodes is unknown to the algorithms. Thus, we are free to choose values for the Y^i, consistent with the global bounds, without affecting either algorithm's decision to evaluate or prune X^0. We are also free to choose a value for X^0 as well, since the decision to evaluate or prune it is made before its value is known. The rest of the proof consists of carefully choosing values for the Y^i and X^0 so that the value of X^i, for all i, will depend on the value of X^0. This will be done by induction on i.

Basis step: $i = 0$. Trivially, if $i = 0$, then the value of $X^i = X^0$ depends on X^0. However, to strengthen the induction hypothesis, we will choose two different values for X^0. In one case, let $X^0_{p(1)} = 0$, and in the other case, let $X^0_{p(1)} = \alpha_1 + 2\varepsilon$, and $X^0_{p(2)} = 0$, where ε is an arbitrarily small positive value. In particular, choose ε such that $\alpha_i + \alpha_{i-1} + 3\varepsilon < 1$, for all $i \geq 1$.

Induction step. Now assume that for all values up to and including $i - 1$, $X^{i-1}_{p(i)} = 0$, or $X^{i-1}_{p(i)} = \alpha_i + 2\varepsilon$ and $X^{i-1}_{p(i+1)} = 0$, depending on the value of X^0.

Choose the value of Y^{i-1} as follows: $Y^{i-1}_{p(i)} = \alpha_i + \varepsilon$ and $Y^{i-1}_{p(i+1)} = \alpha_{i+1} + 2\varepsilon$ and $Y^{i-1}_{p(i+2)} = 0$. Note that the sum of the Y^{i-1} components is $\alpha_i + \varepsilon + \alpha_{i+1} + 2\varepsilon + 0 = \alpha_i + \alpha_{i+1} + 3\varepsilon < 1$, since $\alpha_i + \alpha_{i+1} < 1$, and ε was chosen to satisfy this property. Thus, this is a legal set of components for the value of Y^{i-1}.

Now there are two cases to consider: In case I, $X^{i-1}_{p(i)} = 0$. In that case, since $Y^{i-1}_{p(i)} = \alpha_i + \varepsilon > \alpha_i \geq 0 = X^{i-1}_{p(i)}$, Y^{i-1} is the best child of X^i, and the value of X^i equals the value of Y^{i-1}. In particular, $X^i_{p(i+1)} = Y^{i-1}_{p(i+1)} = \alpha_{i+1} + 2\varepsilon$ and $X^i_{p(i+2)} = Y^{i-1}_{p(i+2)} = 0$. In case II, $X^{i-1}_{p(i)} = \alpha_i + 2\varepsilon$ and $X^{i-1}_{p(i+1)} = 0$. In that case, since $X^{i-1}_{p(i)} = \alpha_i + 2\varepsilon > \alpha_i + \varepsilon = Y^{i-1}_{p(i)} > \alpha_i$, X^{i-1} is the best child of X^i, and the value of X^i equals the value of X^{i-1}. In particular, $X^i_{p(i+1)} = X^{i-1}_{p(i+1)} = 0$. Thus, in one case, $X^i_{p(i+1)} = 0$, and in the other case, $X^i_{p(i+1)} = \alpha_{i+1} + 2\varepsilon$ and $X^i_{p(i+2)} = 0$. This is the induction hypothesis for i.

Thus, the induction hypothesis is true for i, given that it is true for $i-1$. Therefore, for such a tree of any height, the maxn value of the root depends on X^0. But if algorithm A prunes X^0, it cannot determine the maxn value of the root. This contradicts our assumption that algorithm A works for any maxn tree. Therefore, every directional algorithm that computes the maxn value of a game tree with more than two players must evaluate every terminal node evaluated by shallow pruning. □

3.6. *Best-case performance*

How effective is shallow pruning in the best case? For simplicity, we will exclude immediate pruning by assuming that no one component can equal the upper bound on the sum. The best-case analysis of shallow pruning is independent of the number of players and was done by Knuth and Moore [1] for two players.

In order to evaluate a node in the best case, one child must be evaluated, and then evaluating one grandchild of each remaining child will cause the remaining grandchildren to be pruned (see Fig. 3). Thus, If $F(d)$ is the number of leaf nodes generated to evaluate a tree of depth d with branching factor b in the best case, then $F(d) = F(d-1) + (b-1)*F(d-2)$. Since a tree of depth zero is a single node, and a tree of depth one requires all children to be evaluated, the initial conditions are $F(0) = 1$ and $F(1) = b$. Note that in a binary tree, $F(d)$ is the familiar Fibonacci sequence. The solution to the general recurrence has an asymptotic branching factor of $\frac{1}{2}(1 + \sqrt{4b-3})$. For large values of b, this approaches $\sqrt{b}$ which is the best-case performance of full two-player alpha-beta pruning.

3.7. *Average-case performance*

Knuth and Moore [1] also determined that in the average case, the asymptotic branching factor of two-player shallow pruning is approximately $b/\log b$. They assumed independent, distinct leaf values.

In the case of multiple players, however, our model of the evaluation function must have a lower bound on each component and an upper bound on their sum. For simplicity, assume that the lower bound is zero and that the sum is exactly one. Thus, we need a way of randomly choosing n-tuples such that each component is identically distributed between zero and one, and the sum of all components is one. One way to do this is by cutting the zero-one interval in $n-1$ places, with each cut point independently and identically distributed from zero to one, and using the n resulting segments as the components of the n-tuple. Another way is to choose n values independently and identically distributed from zero to one, and then divide each component by the sum of all of them. Furthermore, we assume that each tuple is generated independently of the others.

Under this average-case model, the asymptotic branching factor of shallow pruning with more than two players is simply b, the brute-force branching factor. The analysis relies on the minimax convergence theorem [4], which holds for two-player minimax trees. This surprising phenomenon is that if the leaf values are chosen independently from the same distribution, the variance of the root value decreases with increasing height of the tree, and in the limit of infinite height, the root value can be predicted with probability one. The actual limiting value depends on the leaf distribution and also on which player moves last in the tree, but the convergence does not. Based on empirical studies, we conjecture that minimax convergence applies to maxn trees as well.

In order for pruning to take place, the lower bound on one component must be greater than or equal to its upper bound, which equals one minus the lower bound on another component. Thus, pruning only takes place when the sum of the lower bounds on two different components is greater than or equal to one. In order for this to occur in the limiting value, the values of the remaining components must be zero, since the sum of the two components in question is one. This cannot happen in the limiting value, assuming continuous terminal values. Thus, while pruning occurs at low levels of the tree, at higher levels it becomes increasingly rare, and in the limit of infinite depth, it disappears entirely. Thus, the asymptotic branching factor is simply b.

The shallow pruning algorithm has been implemented and its efficiency tested using the above model of independent and identically distributed cut points. Not only does the effective branching factor converge to b, but the convergence is rapid enough that even for small trees, the branching factor is very close to b. Thus, in practice, alpha-beta is not effective for multi-player trees.

In the case of two-player alpha-beta, the performance of the algorithm is purely a function of the order of the terminal values and not the values themselves. In particular, Knuth and Moore [1] show that for any set of terminal values there is an ordering for which alpha-beta achieves its best-case performance. For example, if all terminal values were equal, then alpha-beta

evaluates only the minimum number of nodes. In the multi-player case, the situation is quite different. While node ordering can have an effect on the performance of multi-player pruning, even under optimal ordering the actual values can limit the effectiveness of the algorithm. In particular, if at a given level of the tree all the values are identical and more than two components are non-zero, which is what happens given maxn convergence, then no further pruning is possible above that level in the tree. Thus, the above result, that shallow pruning does not reduce the asymptotic branching factor in the average case, does not depend on the ordering of nodes in the tree.

4. Optimality of alpha-beta

Since Theorem 1 shows that shallow pruning is an optimal directional algorithm for game trees with more than two players, an obvious question is what is the optimal directional algorithm for two-player minimax trees? The answer is standard alpha-beta pruning. While far from surprising, this result has not appeared in the literature, to our knowledge.

Theorem 3. *Every directional minimax search algorithm must evaluate every leaf node evaluated by alpha-beta under the same ordering.*

Proof. Assume the converse. Namely, that there exists a directional minimax algorithm A, and a minimax tree T, with a leaf node n that is evaluated by alpha-beta but not by algorithm A. Since both algorithms are directional, (re)order the nodes in T from left to right in the order they are visited. Consider the state of the alpha-beta algorithm just before it evaluates node n. It consists of the path from the root to node n, together with a set of lower bounds α_i on the MAX ancestors of n, and a set of upper bounds β_i on the MIN ancestors of n. When the algorithm is called on node n, α is the maximum of the α_i, or the greatest lower bound on all of the MAX ancestors of n, and β is the minimum of the β_i, or the least upper bound on all of the MIN ancestors of n, based on all nodes to the left of node n. Since by assumption node n is evaluated by alpha-beta, α must be strictly less than β, or node n would have been pruned. Now, construct a new tree T' from T by removing all branches to the right of the path from the root to node n, making node n the last frontier node in a left-to-right traversal of T'. Choose a value x for node n in T' that is greater than α and less than β. Since x is strictly greater than the lower bound on all its MAX ancestors in T', and strictly less than the upper bound on all its MIN ancestors in T', x is the unique minimax value of tree T'. Since algorithm A by assumption is a directional algorithm and decides to prune node n in T based on the nodes to its left, it must also prune node n in T' because T and T' are identical to the left of node n. But then algorithm A cannot correctly

compute the minimax value of T', since it is uniquely determined by node n which is pruned by A in T'. This contradicts our assumption that algorithm A is a correct minimax search algorithm. Therefore, every directional minimax algorithm must evaluate every leaf node evaluated by alpha-beta. □

5. Conclusions

We considered the extension of alpha-beta pruning to games with more than two players. Minimax was generalized to maxn by [2]. If we assume that there is a lower bound on each component of the evaluation function, and an upper bound on the sum of all components, then shallow alpha-beta pruning is valid, but not deep pruning. In the best case, this results in significant savings in computation, but in the average case it does not reduce the asymptotic branching factor. This implies that the effectiveness of alpha-beta is limited to the case of two players. We also showed that alpha-beta is an optimal directional algorithm for two-player games.

Acknowledgement

This research was supported by an NSF Presidential Young Investigator Award, and NSF Grant IRI-8801939. Thanks to Chris Ferguson and Judea Pearl for helpful discussions concerning this work, to the anonymous reviewers for their comments, and to Valerie Aylett for drawing the figures.

References

[1] D.E. Knuth and R.E. Moore, An analysis of alpha-beta pruning, *Artif. Intell.* **6** (1975) 293–326.

[2] C.A. Luckhardt and K.B. Irani, An algorithmic solution of N-person games, in: *Proceedings AAAI-86*, Philadelphia, PA (1986) 158–162.

[3] A. Newell, J.C. Shaw and H.A. Simon, Chess playing programs and the problem of complexity, *IBM J. Res. Dev.* **3** (1959) 320–335. Reprinted in: E.A. Feigenbaum and J. Feldman, eds., *Computers and Thought* (McGraw-Hill, New York, 1963) 39–70.

[4] J. Pearl, *Heuristics* (Addison-Wesley, Reading, MA, 1984).

[5] C.E. Shannon, Programming a computer for playing chess, *Philos. Mag.* **41** (1950) 256–275.