On Pruning Techniques for Multi-Player Games

Nathan R. Sturtevant and Richard E. Korf
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024
{nathanst, korf}@cs.ucla.edu

Abstract

Max^n (Luckhardt and Irani, 1986) is the extension of the minimax backup rule to multi-player games. We have shown that only a limited version of alpha-beta pruning, shallow pruning, can be applied to a max^n search tree. We extend this work by calculating the exact bounds needed to use this pruning technique. In addition, we show that branch-and-bound pruning, using a monotonic heuristic, has the same limitations as alpha-beta pruning in a max^n tree. We present a hybrid of these algorithms, alpha-beta branch-and-bound pruning, which combines a monotonic heuristic and backed-up values to prune even more effectively. We also briefly discuss the reduction of a *n*-player game to a 'paranoid' 2-player game. In Sergeant Major, a 3-player card game, we averaged node expansions over 200 height 15 trees. Shallow pruning and branch-and-bound each reduced node expansions by a factor of about 100. Alpha-beta branch-and-bound reduced the expansions by an additional factor of 19. The 2-player reduction was a factor of 3 better than alpha-beta branch-and-bound. Using heuristic bounds in the 2-player reduction reduced node expansions another factor of 12.

Introduction and Overview

Much work and attention has been focused on two-player games and alpha-beta minimax search (Knuth, Moore, 1975). This is the fundamental technique used by computers to play at the championship level in games such as chess and checkers. Alpha-beta pruning works particularly well on games of two players, or games with two teams, such as bridge. Much less work has been focused on games with three or more teams or players, such as Hearts. In max^n (Luckhardt and Irani, 1986), the extension of minimax to multi-player games, pruning is not as successful.

This paper focus on pruning techniques. There are many open questions in multi-player games, and we cannot cover them all here. For instance, it is unclear what the 'best' practical backup rule is. The techniques presented in this paper represent just one way we can evaluate the effectiveness of an algorithm.

We first review the max^n algorithm and the conditions under which pruning can be applied to max^n. Based on this, we show that shallow pruning in max^n cannot occur in many multi-player games. We will examine another common pruning method, branch-and-bound pruning, showing that it faces the same limitations as alpha-beta pruning when applied to max^n trees. Finally, we present a hybrid algorithm, alpha-beta branch-and-bound, which combines these two pruning techniques in multi-player games for more effective pruning. We will also analyze the reduction of a *n*-player game to a 2-player game.

Examples: Hearts and Sergeant Major (8-5-3)

To help make the concepts in this paper more clear, we chose two card games, Hearts and Sergeant Major, to highlight the successes and failures of the various algorithms presented. Note that while the game of bridge is played with 4 players, each player has the goal of maximizing the joint score they share with their partner, so bridge is really a two-team game, and standard minimax applies.

Hearts and Sergeant Major, also known as 8-5-3, are both trick-based card games. That is, the first player plays (leads) a card face-up on the table, and the other players follow in order, playing the same suit if possible. When all players have played, the player who played the highest card in the suit that was led "wins" or "takes" the trick. He then places the played cards in his discard pile, and leads the next trick. This continues until all cards have been played. Cards are dealt out to each player before the game begins, and each game has special rules about passing cards between players before starting. Card passing has no bearing on the work presented here, so we ignore it.

Hearts is usually played with four players, but there are variations for playing with three or more players. The goal of Hearts is to take as few points as possible. Each card in the suit of hearts is worth one point, and the queen of spades is worth 13. A player takes points when he takes a trick which contains point cards. At the end of the game, the sum of all scores is always 26, and each player can score between 0 and 26. If a player takes 26 points, or "shoots the moon," the other players all get 26 points each. For now, we ignore this rule.

Sergeant Major is a three-player game. Each player is dealt 16 cards, and the remainder of the deck is set aside. The ultimate goal for each player is to take as many tricks as possible. Similar to Hearts, the sum of scores is always 16, and each individual player can get any score from 0 to 16.

More in-depth descriptions of these and other games mentioned here can be found in (Hoyle et al. 1991).

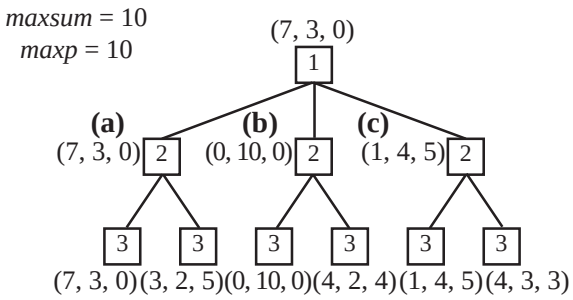

Figure 1: A 3-player max^n game tree.

Max^n

Luckhardt's and Irani's extension of minimax for multi-player games is called max^n. For a n-player game, an n-tuple of scores records each player's individual score for that particular game state. That is, the nth element in the tuple represents the score of the nth player. At each node in a max^n search tree, the player to move selects the move that maximizes his own component of the score. The entire tuple is backed up as the max^n value of that node. In a three-player game, we propagate triples from the leaves of the tree up to the root.

For example, in Figure 1, the triples on the leaves are the terminal values of the tree. The number inside each square represents the player to move at that node. At the node labelled (a), Player 2 will choose to back up the triple (7, 3, 0) from the left child, because the second component of the left child of (a), 3, is greater than the second component of the right child of (a), 2. Player 2 does likewise at nodes (b) and (c). Player 1 then chooses a triple from those backed up by Player 2. At the root, the first component of Player 1's children is greatest at node (a). Player 1 will back this triple up, giving the final max^n value of the tree, (7, 3, 0). Because the max^n value is calculated in a left-to-right depth-first order, a partial bound on the max^n value of a node is available before the entire calculation is complete. Throughout this paper we assume that nodes are generated from left to right in the tree, and that all ties are broken to the left.

When generating a Hearts game tree, the terminal values will be the number of points taken in the game. In Sergeant Major, the terminal values will be the number of tricks taken. If we are not able to search to the end of the game, we can apply an evaluation function at the frontier nodes to generate appropriate backup values. At a minimum, this evaluation would be the exact evaluation of what has occurred so far in the game, but might also contain an estimate of what scores are expected in the remainder of the game.

In most card games, one is not normally allowed to see one's opponents cards. As was suggested by (Ginsberg, 1996), we first concentrate on being able to play a completely open (double-dummy) game where all cards are available for all to see. In a real game, we would model the probability of our opponent holding any given card, and then generate hundreds of random hands according to these probability models. It is expected that solving these hands will give a good indication of which card should actually be played. See (Ginsberg, 1999) for an explanation of how this has been applied to Bridge.

Duality of Maximization and Minimization

Throughout this paper we deal with games that are usually described in terms of either maximization or minimization. Since minimization and maximization are symmetric, we briefly present here how the bounds used by pruning algorithms are transformed when we switch from one type of game to the other type.

There are four values we can use to describe the bounds on players' scores in a game. *Minp* and *maxp* are a player's respective minimum and maximum possible score. *Minsum* and *maxsum* are the respective minimum and maximum possible sum of all players scores. In Hearts, *minp* is 0 and *maxp* = *maxsum* = *minsum* = 26. In Sergeant Major, *minp* is also 0 and *maxp* = *maxsum* = *minsum* = 16. (Korf, 1991) showed that we may be able to prune a max^n tree if *minp* and *maxsum* are bounded. We are interested in how these bounds change when the goal of a game is changed from minimization to maximization. The transformation does not change the properties of the game, it simply allows us to talk about games in their maximization forms without loss of generality.

The one-to-one mapping between the minimization and maximization versions of a game is shown in Table 1. The first row in the table contains the variable names for a minimization problem, followed by sample values for a Hearts game, where *n*, the number of players, is 3. The transformation applied to the values are in the third row: the negation of the original value plus $maxp_{min}$. This re-normalizes the scores so that *minp* is always 0. Since Hearts and Sergeant Major are zero-sum or constant-sum games, *maxsum* is always the same as *minsum*. The final rows contain the new score after transformation and the new variable names. The process can be reversed to turn a maximization game into a minimization game.

Given the symmetry of minimization and maximization, there is also a duality in pruning algorithms. That is, for any pruning algorithm that works on a maximization tree, we can write the dual of that algorithm that works the same under the equivalent minimization tree. However, just changing the goal of a game from minimization to maximization does not create the dual of the game. The other parameter,

minimization variable	s_1	s_2	s_3	$maxp_{min}$	$minp_{min}$	$maxsum_{min}$ & $minsum_{min}$
minimization value	3	10	13	26	0	26
transformation		$-s_i + maxp_{min}$		$-maxp_{min} + maxp_{min}$	$-minp_{min} + maxp_{min}$	$-maxsum_{min} + n \cdot maxp_{min}$
maximization value	23	16	13	0	26	52
maximization variable	s^1	s^2	s^3	$minp_{max}$	$maxp_{max}$	$maxsum_{max}$ & $minsum_{max}$

Table 1: The transformation between a maximization and minimization problem, and examples for a 3-player game.

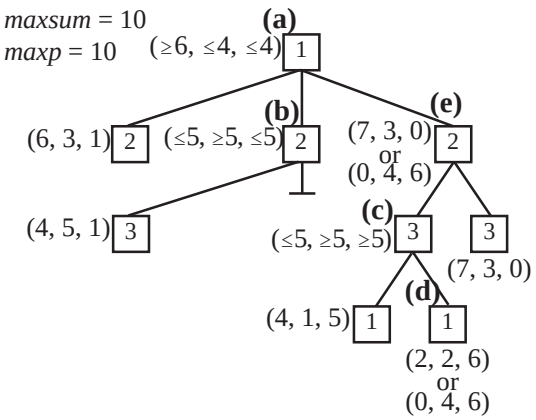

Figure 2: Pruning in a max^n tree.

maxsum, must also be calculated. Given these observations, we will not explicitly show dual algorithms. Unless otherwise stated, all trees and algorithms presented here will be for maximization problems.

Pruning in Max^n Trees

In a two-player zero-sum game, there are three types of alpha-beta pruning that occur: immediate, shallow, and deep pruning. Not all of these are valid in multi-player games.

Immediate Pruning

Immediate pruning in a multi-player game is like immediate pruning in a two-player game. In a two-player game, we immediately prune when a player gets the best possible score, ∞ for max and -∞ for min. In a multi-player game, we can prune when the current player gets a score of *maxp*, the best score in a multi-player game.

The opportunity to prune immediately is seen in Figure 1. At node (b), Player 2 can get 10 points by choosing to move towards his left child. Since *maxp* = 10, Player 2 can do no better than 10 points. Thus, after examining the first child, the second child can be pruned.

Shallow Pruning

While having a zero-sum game is a sufficient condition to apply alpha-beta pruning to a two-player game tree, it is not sufficient for a multi-player game tree. Given just one component of a zero-sum multi-player game, we cannot restrict any other single score in the game, because one of the remaining scores might be arbitrarily large, and another arbitrarily small. But, given a lower bound on each individual score, and an upper bound on the sum of scores, we can prune.

Figure 2 contains a sample 3-player max^n tree. At node (a), Player 1 can get at least 6 points by choosing the left-most branch of node (a). When Player 2 examines the first child of node (b), Player 2 gets a score of 5, meaning Player 2 will get at least 5 by choosing the left-most branch at (b). There are 10 points available in the game, and since Player 2 will get at least 5 at node (b), Player 1 can get no more than 10 - 5 = 5 points at (b). Player 1 is guaranteed ≥6 points at (a), and ≤5 points at (b). So, Player 1 will never move towards node (b) no matter what max^n values the other children have, and the remaining children of (b) are pruned. This is shallow pruning, because the bound used to prune came from (a), the parent of (b).

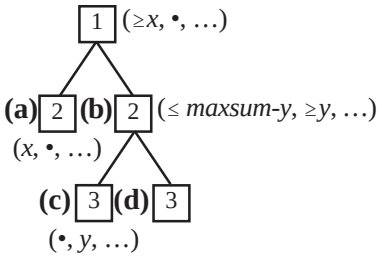

Figure 3: A generic max^n tree.

General Bounds for Shallow Max^n Pruning

Figure 3 shows a generic max^n tree. In this Figure we have only included the values needed for shallow pruning. Other values are marked by a '•'. When Player 1 gets a score of x at node (a), the lower bound on Player 1's score at the root is then x. Assume Player 2 gets a score of y at node (c). Player 2 will then have a lower bound of y at node (b). Because of the upper bound of *maxsum* on the sum of scores, Player 1 is guaranteed less than or equal to *maxsum* - y at node (b). Thus, no matter what value is at (d), if *maxsum* - $y \le x$, Player 1 will not choose to move towards node (b) because he can always do no worse by moving to node (a), and we can prune the remaining children of node (b).

In the maximization version of Hearts, *maxsum* is 52, and x and y will range between 0 and 26, meaning that we only prune when 52 - $y \le x$, which is only possible if $x = y = 26$. In Sergeant Major *maxsum* is 16, and x and y will range from 0 to 16, meaning that we will prune when 16 - $y \le x$.

Given these examples, we extract general conditions for pruning in multi-player games. We will use the following variables: n is the number of players in the game, *maxsum* is the upper bound on the sum of players scores, and *maxp* is the upper bound on any given players score. We assume a lower bound of zero on each score without loss of generality. So, by definition, $maxp \le maxsum \le n \cdot maxp$.

Lemma 1:
To shallow prune in a max^n tree, $maxsum < 2 \cdot maxp$.

Proof:
We will use the generic tree of Figure 3. To prune:
$x \ge maxsum - y$
By definition:
$2 \cdot maxp \ge x + y$
So,
$2 \cdot maxp \ge x + y \ge maxsum$
$2 \cdot maxp \ge maxsum$

However, if $maxsum = 2 \cdot maxp$, we can only prune when both x and y equal *maxp*. But, if $y = maxp$, we can also im-

mediate prune. Because of this, we tighten the bound to exclude this case, and the lemma holds. □

We can now verify what we suggested before. In the maximization version of 3-player Hearts, *maxsum* = 52, and *maxp* = 26. Since the strict inequality of Lemma 1, $52 < 2 \cdot 26$, does not hold, we can only immediate prune in Hearts. In Sergeant Major, the inequality $16 < 2 \cdot 16$ does hold, so we will be able to shallow prune a Sergeant Major max^n tree.

Intuitive Approach. Speaking in terms of the games as they are normally played, it may seem odd that we can't prune in Hearts and we can prune in Sergeant Major, when the only real difference in the games is that it one you try to minimize your score, and in the other you try to maximize it. While the preceding lemma explains the difference mathematically, there is another explanation that may be more intuitive.

Suppose in Sergeant Major that a player is deciding between two cards, the Ace of Spades and the Ten of Clubs. When we calculate the max^n value of the search tree, we are calculating how well the player can expect to do when playing a given card. Once we have the result of how well the player can do with the Ace of Spades, we begin to look at the prospects for the Ten of Clubs. We prune this search when we have enough information to guarantee that the player will always do no better with the Ten of Clubs than with the Ace of Spades. We get this information based on the dependence between the players' scores. In Sergeant Major, there are only 16 points available, and all players are competing to get as many points as possible. Each trick taken by one player is a trick denied to another player. This direct dependence between any two players score is what gives us the information that allows us to prune. When the next player is guaranteed enough points to deny a better score than can be achieved by playing the Ace of Spades, the line of play originating from the Ten of Clubs is pruned.

In the standard minimization form of Hearts, the goal is to take as few points as possible. Points taken by one player are points denied to the other players. But, since all players are trying to take as few points as possible, they don't mind being denied points. Thus, when another player takes points, it simply tells us that the current line of play may be better than previous lines of play, and that we should keep exploring our current line of play. When one player avoids taking points, those points must be taken by the other players. But, there is nothing that says which player must take the points. So, in contrast to Sergeant Major, there is a lack of direct dependence between two players scores, and we are unable to prune.

Deep Pruning

Returning to Figure 2, Player 1 is guaranteed a score greater than or equal to 6 at the root node (a). We might be tempted to prune node (d), because the bound on Player 1's score at (c), ≥ 5, says that Player 1 will get less than 6 points. This would be deep pruning, because (a) is a grandparent of (c). However, as we demonstrate here, the value at node (d) can still affect the max^n value of the tree. (Korf 1991)

If the value of (d) is (2, 2, 6), Player 3 will choose this value as the max^n value of (c). Player 2 at (e) will then choose (7, 3, 0) as the max^n value of (e) since the second component, 3, is higher than the second component of the max^n value at (c), 2. This will result in the max^n value of (7, 3, 0) for the entire tree, since Player 1 can then get a score of 7.

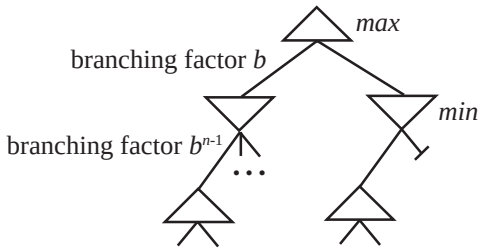

Figure 4: The reduction of a *n*-player game to a 2-player game.

Alternatively, if the value of (d) is (0, 4, 6), the max^n value of (c) will be (0, 4, 6). Then, at node (e), Player 2 will choose to backup (0, 4, 6) because the second component, 4, is higher than that in the other child, 3. This means the final max^n value of the tree will be (6, 3, 1).

Thus, while the bounds predicted correctly that no value at (d) will ever be the final max^n value of the tree, the different possible values at (d) may affect the final max^n value of the tree, and so (d) cannot be pruned.

Asymptotic Results

The asymptotic branching factor of max^n with shallow pruning in the best case is $\left(1+\sqrt{4b-3}\right)/2$, where b is the brute-force branching factor without any pruning. An average case model predicts that even under shallow pruning, the asymptotic branching factor will be b. (Korf, 1991)

We have shown here that in many cases, such as the game of Hearts, even under an optimal ordering of the tree, we would still be unable to do anything besides immediate pruning. This compares poorly with the 2-player best-case asymptotic branching factor of $\sqrt{b}$ (Knuth, Moore 1975), which can very nearly be achieved in two-player games.

Reduction to a Paranoid 2-Player Game

Another method to increase the pruning in a multi-player game is to reduce the game to a two-player game. This is done by making the 'paranoid' assumption that all our opponents have formed a coalition against us. Under this reduction we can use standard alpha-beta to prune our tree. This is not a realistic assumption and can lead to suboptimal play, but due to the pruning allowed, it may be worthwhile to examine. We will only analyze the pruning potential here.

To calculate the minimum number of nodes that need to be examined within the game tree, we need a strategy for min and a strategy for max. Min and max will play on the tree in Figure 4, where max is to move at the root, with a branching factor of b, and min moves next, with a branching factor of b^{n-1}. Min is the combination of the n-1 players playing against the first player.

Within a strategy for max, max must look at one successor of each max node in the strategy, and all possible successors of each min node in the strategy. Suppose the full tree is of depth D. Max will expand $b^{(n-1)}$ nodes at every other level, meaning that there are $b^{(n-1)\cdot D/2}$ leaf nodes in the

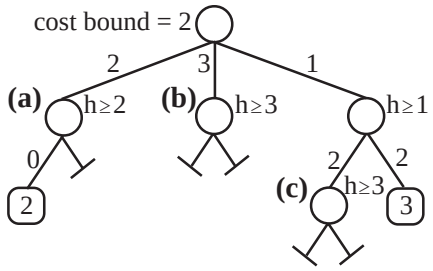

Figure 5: A single-agent depth-first branch-and-bound problem.

tree. Similarly, a min strategy must look at only one successor of each min node, and all successors of each max node, so min will look at $b^{D/2}$ nodes total. We have two players in the reduced game, and each player has an equal number of turns, so D is even, meaning we don't have to consider the floor or ceiling in the exponent.

The total nodes examined by both algorithms will be about $b^{(n-1)\cdot D/2} + b^{D/2}$ nodes, which is $O(b^{(n-1)\cdot D/2})$. But, D is the depth in the tree of Figure 4. We really want our results in terms of the real tree that we will search. For example, if the original tree has 3 players and is depth 12 (4 tricks), the new tree has 2 players and will also contains 4 tricks, so it will be height 8. So, for the actual tree searched, which has height d, $D = d \cdot 2/n$. Thus, we re-write the asymptotic branching factor in the best case as $O(b^{d\cdot(n-1)/n})$ to reflect the branching factor in the actual tree.

Depth-First Branch-and-Bound Pruning

Branch-and-Bound is another common pruning technique. It requires a monotonic heuristic, and many card games have natural monotonic heuristics. In Hearts and Sergeant Major, once you have taken a trick or a point you cannot lose it. Thus, an evaluation can be applied within the tree to give a bound on the points or tricks to be taken by a player in the game. We use the notation $h(i) \geq j$ to indicate that the heuristic is giving a lower bound score of j for player i, and $h(i) \leq j$ to indicate that the heuristic is giving an upper bound of j on player i's score. Suppose, for a Sergeant Major game, the players have taken 3, 2, and 6 points respectively. Then, $h(1) \geq 3$ because Player 1 has taken 3 points. Also, $h(1) \leq 8$ because *maxsum* (16) minus the other players' scores (8) is 8.

Single Agent Branch-and-Bound

The branch-and-bound algorithm is most commonly used in a depth-first search to prune single-agent minimization search trees, such as the Travelling Salesman Problem. In Figure 5, we are trying to find the shortest path to a leaf from the root, where edges have positive costs as labelled. Since all paths have positive length, the cost along a path will monotonically increase, giving a lower bound on the cost to a leaf along that path. The labels at the leaves are the actual path costs. Next to a node is a limit on the optimal cost of a path going through that node. If unexplored paths through a node are guaranteed to be greater than the best path found so far, we can prune the children of that node in the tree.

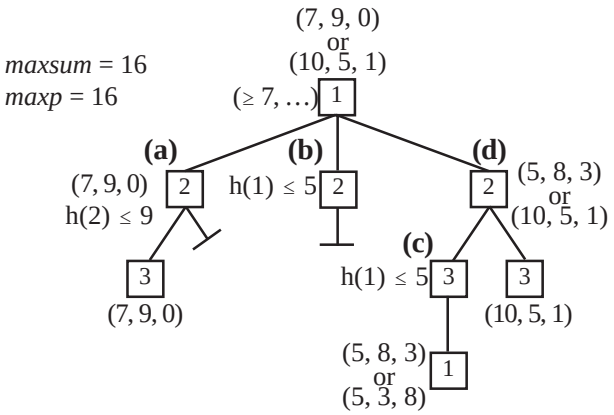

Figure 6: Branch-and-bound pruning in a max^n tree.

In order to draw parallels between alpha-beta pruning, we will describe the pruning that occurs in the same terms that we use to describe alpha-beta pruning: immediate, shallow and deep pruning. In a two-player game, immediate pruning occurs when we get the best score possible, a win. In the presence of a heuristic, the best score possible is best that we can get given the heuristic. In Figure 5, the heuristic at node (a) says the best score we can get is 2. Since we have a path of total cost 2 through the first child, we can prune the remaining children, as we have found the best possible path.

After finding the path with cost 2, we use that cost as a bound while searching subsequent children. At node (b), our heuristic tells us that all paths through (b) have cost higher than the bound of 2, so all children of (b) are pruned. This is like shallow pruning, since the bound comes from the parent of (b). Finally, at node (c) we can prune based on the bound of 2 on the path cost from the grandparent of (c), which is like deep pruning.

Multi-Player Branch-and-Bound

Branch-and-bound pruning can be used to prune a max^n tree, but under max^n it is limited by the same factors as alpha-beta pruning, namely we cannot use the bound at a node to prune at its grandchild. As with deep alpha-beta pruning, while the max^n value of the pruned nodes will never be the max^n value of the tree, they still have the potential to affect it. We will demonstrate this here, but because the proof is identical to the proof of why deep alpha-beta pruning does not work (Korf, 1991), we omit the proof.

In Figure 6 we show a portion of a max^n tree and demonstrates how branch-and-bound can prune parts of the tree. Immediate pruning occurs at node (a). At the left child of (a), Player 2 can get a score of 9. Since the $h(2) \leq 9$, we know Player 2 cannot get a better score from another child, and the remaining children are pruned.

Shallow pruning occurs at node (b) when the bound from the parent combines with the heuristic to prune the children of (b). Player 1 is guaranteed 7 or more at the root. So, when Player 1's heuristic at (b) guarantees a score of 5 or less, we prune all the children of (b), since Player 1 can always do better by moving to node (a).

Finally, deep branch-and-bound pruning, like deep alpha-

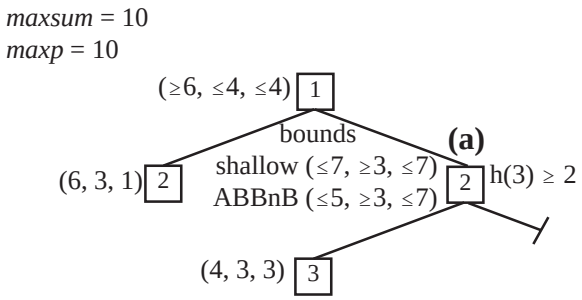

Figure 7: Alpha-beta branch-and-bound pruning.

beta pruning, can incorrectly affect the calculation of the max^n value of the game tree. The partial max^n value at the root of the tree in Figure 6 guarantees Player 1 a score of 7 or better. At node (c), Player 1 is guaranteed less than or equal to 5 points by the heuristic. Thus, we might be tempted to prune the children of (c), since Player 1 can do better by moving to node (a). But, this reasoning does not take into account the actions of Player 2.

Depending on which value we place at the child of (c), (5, 8, 3) or (5, 3, 8), Player 2 will either select (5, 8, 3) from node (c) or (10, 5, 1) from node (d)'s right branch to back up as the max^n value of node (d). Player 1 would then choose the root max^n value to be either (7, 9, 0) or (10, 5, 1). So, while the bounds on node (c) will keep it from being the max^n value of the tree, it has the potential to affect the max^n value of the tree.

Alpha-Beta Branch-and-Bound Pruning

Now that we have two relatively independent techniques for pruning a multi-player game tree, we show how these techniques can be combined. Shallow pruning makes comparisons between two players' backed up scores to prune. Branch-and-bound compares a monotonic heuristic to a player's score to prune. Alpha-beta branch-and-bound pruning uses both the comparison between backed up scores and monotonic heuristic limits on scores to prune even more effectively.

Looking at Figure 7, we see an example where shallow pruning applies. We have bounds on the root value of the tree from its left branch. After searching the left child of node (a) we get bounds on the max^n value of (a). We place an upper bound of 7 on Player 1's score, because Player 2 is guaranteed at least 3 points, and 10 (*maxsum*) - 3 = 7. This value does not conflict with the partial max^n bound on the root, so we cannot prune. We have a bound from our heuristic, but because it is not Player 3's turn, we can not use that by itself to prune either. But, if we combine this information, we can tighten our bounds. We know from backed up values that Player 2 will get at least 3 points and from our heuristic that Player 3 will get at least 2 points at (a). So, the real bound on Player 1's score is *maxsum* - score(2) - h(3) = 10 - 3 - 2 = 5.

As an aside, one may notice another slight, but effective optimization in this example. At (a), Player 2 will not choose another path unless he gets at least 4 points, and thus Player 1 gets no more than 6. Thus, since ties are broken to the left, we have integer terminal values, and because Player 1 did not get 7 points at the left child of (a), the shallow bound itself is sufficient to prune the right branch of (a).

In a *n*-player game where we normally only compare the scores of two players, we can further decrease our bound for pruning by subtracting the heuristic value for the remaining (*n* - 2) players. That is, if we have a lower bound on Player *i*'s score from our parent, and Player *j* is to play at the current node, the upper bound on Player *i*'s score at the next node is *maxsum* - score(*j*) - $\sum h(x)$ {for $x \neq i$ or *j*}. In a two-player game, this reduces to plain alpha-beta.

The alpha-beta branch-and-bound procedure is as follows. In this procedure, we use h_{up} to represent a heuristic upper bound and h_{low} to represent a heuristic lower bound. *Bound* is the upper bound on *Player*'s score.

```
ABBnB(Node, Player, Bound)
IF Node is terminal, RETURN static value
/* shallow branch-and-bound pruning */
IF (h_up(Prev Player) ≤ maxsum - Bound)
   RETURN static value
Best=ABBnB(first Child, next Player, maxsum)
/* Calculate our opponents guaranteed points */
Heuristic = Σh_low(n) [n≠Player or prev. Player]
FOR each remaining Child
  IF (Best[Player] ≥ Bound-Heuristic) OR
     (Best[Player] = h_up(Player))
       RETURN Best
  Current = ABBnB(next Child, next Player,
                  maxsum - Best[Player])
  IF (Current[Player] > Best[Player])
       Best = Current
RETURN Best
```

This procedure will always prune as much as shallow branch-and-bound pruning or shallow alpha-beta pruning. So, while we lose the ability to do deep pruning in a multi-player game, we may be able to use alpha-beta branch-and-bound pruning to prune more than we would be able to with just alpha-beta or branch-and-bound pruning alone.

Disregarding immediate branch-and-bound pruning, Alpha-beta branch-and-bound will have the same best-case performance as shallow pruning. If we have perfect ordering and a perfect heuristic, immediate branch-and-bound pruning could drastically shrink the search tree.

Experimental Results

We tested alpha-beta branch-and-bound (ABBnB) to see how it compared to branch-and-bound (BnB), alpha-beta shallow pruning, and the paranoid 2-player reduction. Our test domain was the game of Sergeant Major, and our heuristic was the number of tricks taken so far in the game. We searched 200 random game trees to a depth of 5 tricks, which is 15 cards. Consecutive cards in a player's hand were generated as a single successor. Moves were ordered from high cards to low cards. We initially did not use a transposition table or any other techniques to speed the search. Our code expands about 150k nodes per second on a Pentium II 233 laptop,

Algorithm	Full Tree	DFBnB	Shallow	ABBnB	Paranoid	Paranoid (with heuristic)
Avg. Nodes in Tree	3.33 billion	32.7 million	26.8 million	1.43 million	437,600	36,121
Reduction factor	1	102	1.22	18.7	3.27	12.1

Table 2: The average nodes expanded of the first 5 tricks in Sergeant Major and reduction factor over the next best algorithm.

depending on the problem.

The number of nodes in the entire tree varied from 78 million to 64 billion, with the average tree containing 33 billion nodes. The number of nodes expanded by each of the algorithms varied widely, based on the difficulty of the hand. Because of this, we have chosen to report our results according to the average number of nodes expanded by an algorithm over all 200 trees. These results are found in Table 2.

The first line in the table contains the average number of nodes in the entire tree. The second line contains the factor in reduction over the next best algorithm. The algorithms are listed left to right from worst to best. We ran the paranoid algorithm twice, once without using the heuristic information, and once using the heuristic information.

One interesting result is that the shallow pruning procedure provides significant savings over the full tree expansion. Thus, despite the negative theoretical results, there is still some potential for this algorithm.

Another thing to notice is how much faster the paranoid algorithm is than the standard $\max^n$ backup rule. This speed increase will not, however, guarantee an increase in play quality. Under this model, a player may make very poor moves assuming all the other players might work together much more than they really do. Double dummy play can magnify this problem. Clearly more work is needed to distinguish which algorithms are the best to use in practice.

Unfortunately, the most obvious heuristic in Hearts, the points taken by a player so far in the game, will only allow branch-and-bound pruning, and not for alpha-beta branch-and-bound pruning. This is because this heuristic comes directly from the evaluation function, which already doesn't allow shallow pruning. However, a heuristic that came from a different evaluation might allow some pruning.

Conclusion and Future Work

We have refined the bounds needed to prune a $\max^n$ tree using shallow pruning and introduced the alpha-beta branch-and-bound algorithm. While this algorithm is quite effective at reducing the number of nodes expanded in a $\max^n$ tree, it still cannot compare to two-player alpha-beta pruning. A bridge hand can be search in its entirety, but we are not close to doing this in multi-player games such as Sergeant Major, and we are even farther from doing it in Hearts. Alpha-beta branch-and-bound can solve 8-card hands (complete depth 24 trees) to completion in times ranging from a few seconds to about a minute. We are working on a implementation of Partition Search (Ginsberg, 1996) to see how this algorithm benefits searches on deeper trees. Our initial transposition table reduced node expansions by a factor of 3, but also slowed our program by the same factor.

More research needs to be done to see what other algorithms or methods might be applied to help with multi-player search. We are continuing to work to compare the value of these and other algorithms in real play, and as this work progresses we will be evaluating the assumption that we can use double-dummy play to model our opponents hands. It would be worthwhile to develop a different theoretical model to better explain how shallow and alpha-beta branch-and-bound pruning works in practice. Additional work on heuristics and game search can be found in (Prieditis, Fletcher, 1998).

One possibility for improving our search is to use domain specific knowledge for a particular game to simplify the problem. In most trick games, for instance, you must follow suit. This creates a loose independence between suits, which may be exploited to simplify the search process.

Research in practical multi-player game search has been very limited. We expect that in the next few years this will change and that much progress will be made in multi-player game search.

Acknowledgments

We would like to thank the reviewers for their comments and Pamela Allison for early discussion on pruning failures in other games. This work has been supported by the National Science Foundation under grant IRI-9619447.

References

Ginsberg, M, GIB: Steps Toward an Expert-Level Bridge-Playing Program, Proceedings, IJCAI-99, 584-589.

Ginsberg, M, How Computers Will Play Bridge, *The Bridge World*, 1996.

Ginsberg, M, Partition Search, Proceedings AAAI-96, Portland, OR, 228-33.

Hoyle, E., and Frey, R.L., Morehead, A.L., and Mott-Smith, G, 1991, *The Authoritative Guide to the Official Rules of All Popular Games of Skill and Chance*, Doubleday.

Knuth, D.E., and Moore, R.W., An analysis of alpha-beta pruning, Artificial Intelligence, vol. 6 no. 4, 1975, 293-326.

Korf, R.E. Multiplayer alpha-beta pruning. Artificial Intelligence, vol. 48 no. 1, 1991, 99-111.

Luckhardt, C.A., and Irani, K.B., An algorithmic solution of N-person games, Proceedings AAAI-86, Philadelphia, PA, 158-162.

Prieditis, A.E.; Fletcher, E. Two-agent IDA*, Journal of Experimental and Theoretical Artificial Intelligence, vol.10, Taylor & Francis, 1998, 451-85.