Last-Branch and Speculative Pruning Algorithms for Max^n

Nathan Sturtevant
UCLA, Computer Science Department
Los Angeles, CA 90024
nathanst@cs.ucla.edu

Abstract

Previous work in pruning algorithms for max^n multi-player game trees has produced shallow pruning and alpha-beta branch-and-bound pruning. The effectiveness of these algorithms is dependant as much on the range of terminal values found in the game tree as on the ordering of nodes. We introduce last-branch and speculative pruning techniques which can prune any constant-sum multi-player game tree. Their effectiveness depends only on node-ordering within the game tree. As b grows large, these algorithms will, in the best case, reduce the branching factor of a n-player game from b to $b^{(n-1)/n}$. In Chinese Checkers these methods reduce average expansions at depth 6 from 1.2 million to 100k nodes, and in Hearts and Spades they increase the average search depth by 1-3 ply.

1 Introduction

While the minimax algorithm with alpha-beta pruning [Knuth and Moore, 1975] has dominated the study of 2-player games, a clearly dominant algorithm has not emerged in the study of n-player games.

The standard algorithm for n-player games is max^n [Luckhardt and Irani, 1986]. Only some max^n game trees can be pruned using established techniques such as shallow pruning [Korf, 1991] and alpha-beta branch-and-bound pruning [Sturtevant and Korf, 2000]. This is because the amount of pruning possible under these algorithms is dependant on both the node ordering and the range of terminal values found in the game. Common heuristics for games such as Hearts and Chinese Checkers do not have appropriate terminal values for pruning. In two-player games, however, pruning is only dependant on the node ordering in the tree.

We present here last-branch and speculative pruning techniques which can be used to prune any n-player constant-sum max^n game tree. The effectiveness of these algorithms depends only on node ordering within the game tree. Last-branch pruning is similar to a directional algorithm [Pearl, 1984], as it examines the successors of each node left-to-right without returning to previously searched nodes. It does require that you know when you are searching the last branch of a node. Speculative pruning, however, is not a directional algorithm as it may re-search some branches of the tree. Last-branch pruning is a special case of speculative pruning.

The remainder of the paper is as follows. In Section 2 we present a brief overview of sample multi-player domains. In Section 3 we cover previous techniques for pruning max^n trees. Section 4 covers our new techniques, with results from various domains in Section 5.

2 Multi-Player Domains

A multi-player game is one in which there are 3 or more players or teams competing against each other. Many 4-player games are designed for 2 teams, making them two-player games from a theoretical standpoint. We highlight a few of the many interesting multi-player games here.

2.1 Chinese Checkers

One of the best known multi-player board games is Chinese Checkers. This game is played on a board in the shape of a 6-pointed star, with players' pieces starting on opposite sides of the board. The goal is to move your pieces across the board faster than your opponent(s) can. Chinese Checkers can be played with 2-6 players, and is not played in teams.

2.2 Card Games

We use two games here as examples, Spades and Hearts. Spades is usually played with 4 players, but has a 3-player variant. The goal of Spades is to maximize your score. Hearts is usually played with 4 players, but can be played with 3 or more. The goal of Hearts is to minimize your score.

Many card games, including these, are trick-based. Because a trick cannot be given up after being taken, the tricks taken during the game provide a monotonically increasing lower-bound on the final score of the game. In the case of Spades, players get 1 point per trick taken, while in Hearts each player gets 1 point for every heart in the tricks they take, as well as 13 points for the queen of spades. A complete game is made up of multiple hands, where the goal is to minimize or maximize your points over all the hands.

Both Spades and Hearts are imperfect-information

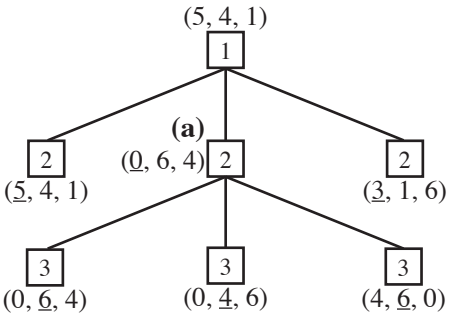

Figure 1. A 3-player max^n tree fragment.

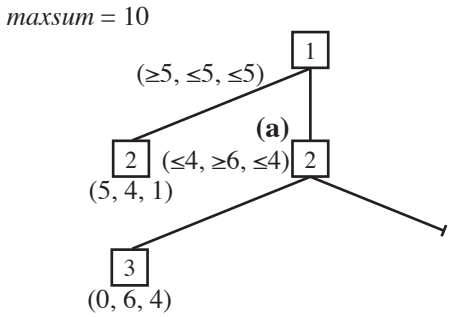

Figure 2. Shallow pruning in a 3-player max^n tree.

games, while the descriptions and trees in this paper assume games have perfect information. In card games, however, we often use perfect-information methods to search imperfect-information games through Monte-Carlo simulation. This involves generating sets of hands that are similar to the hands we expect our opponents to have. We then use traditional perfect-information methods to search these hands, combining and analyzing the results to make the next play. These methods have been successfully used in the Bridge program GIB. [Ginsberg, 2001]

3 Pruning Algorithms for Max^n

The max^n algorithm can be used to play games with any number of players. For two-player games, max^n reduces to minimax. In a max^n tree with n players, the leaves of the tree are n-tuples, where the ith element in the tuple is the ith player's score. At the interior nodes in the game tree, the max^n value of a node where player i is to move is the child of that node for which the ith component is maximum. This can be seen in Figure 1.

In this tree fragment there are three players. The player to move is labeled inside each node. At node (a), Player 2 is to move. Player 2 can get a score of 6 by moving to the left or right and a score of 4 by moving to the middle node. We break ties to the left, so Player 2 will choose the left branch, and the max^n value of node (a) is (0, 6, 4). Player 1 acts similarly at the root selecting the left branch, because the first component of the max^n value there, 5, is greater his score from the middle branch, 0, and his score at the right branch, 3.

3.1 Shallow Pruning

Shallow pruning refers to cases where a bound on a node is used to prune branches from the child of that node. The minimum requirements for pruning a max^n tree with shallow pruning are a lower bound on each players' score and an upper bound, *maxsum*, on the sum of all players' scores. We demonstrate this in Figure 2. In this figure, all scores have a lower bound of 0, and *maxsum* is 10.

Player 1 searches the left branch of the root in a depth-first manner, getting the max^n value (5, 4, 1). Thus, we know that Player 1 will get at least 5 points at the root. Since there are only 10 points available, we know the other players will get less than 5 points at this node.

At node (a), Player 2 searches the left branch to get a score of 6. Because *maxsum* is 10 we know that Player 1's score will never exceed 4 at node (a), so Player 1 will never choose to move towards (a) at the root, and the remaining children of (a) can be pruned.

For shallow pruning to actually occur in a maximization game, each player must have a score that ranges between 0 and *maxsum*. [Sturtevant and Korf, 2000] There are similar bounds for minimization games. In the average case, no asymptotic gain can be expected from shallow pruning. [Korf, 1991] The basic problem with shallow pruning is that it works by comparing the scores of only 2 out of n players in the game, and it is unlikely that 2 players will have their scores sum to *maxsum*. This contrasts with the large gains available from using alpha-beta pruning with 2-player minimax.

3.2 Branch and Bound Pruning

If a monotonic heuristic is present in a game, it can also be used to prune a max^n tree. The full details of how this occurs is contained in [Sturtevant and Korf, 2000]. An example of a monotonic heuristic is the number of tricks taken in Spades. Once a trick has been taken, it cannot be lost. This guarantee can provide a lower bound on a player's score, and an upper bound on one's opponents scores, which can be used in branch-and-bound pruning to prune the game tree. Alpha-beta branch-and-bound pruning combines the actual scores of the 2 players compared in shallow pruning with the heuristic information from the other n-2 players to prune the tree.

3.3 Pruning in Practice

These algorithms have mixed performances in practice. There are no obviously useful monotonic heuristics for Chinese Checkers, and most useful cutoff evaluation functions for the game do not meet the requirements for shallow pruning. Thus, in practice it is not possible to use any of these techniques to prune a Chinese Checkers game tree.

If we use the number of points taken in the game tree so far as the evaluation function, we will be able to use shallow pruning in Spades and branch-and-bound pruning in both Spades and Hearts. This is fine if you can search the entire game tree, but we are currently unable to do that. If we add a predictive component to the evaluation function, these pruning techniques are much less effective. Thus a trade-off has to be made between the quality of the cutoff evaluation function

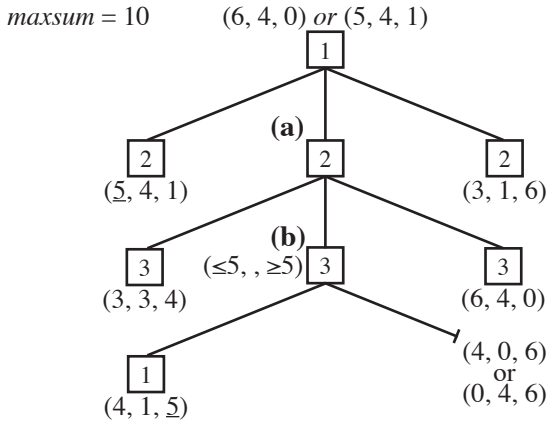

Figure 3: The failure of deep pruning.

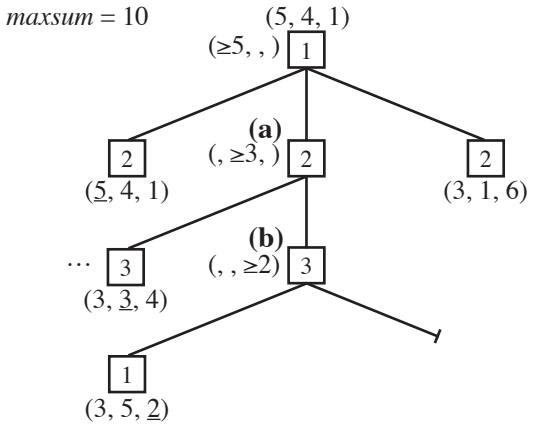

Figure 4. Combining max^n scores to limit value propagation.

and the potential amount of pruning in the game tree.

3.4 Deep Pruning

Deep pruning refers to when the bound at a node is used to prune a grandchild or lower descendant of that node. [Korf, 1991] shows that, in the general case, deep pruning can incorrectly affect the max^n value of the game tree. We demonstrate this in Figure 3. After searching the left branch of node (b), Player 3 is guaranteed 5 points, and Player 1 is guaranteed to get no more than 5 points at node (b). So, we can conclude that the max^n value of node (b) will never be the max^n value of the game tree, because Player 1 is already guaranteed a score of 5 at the root, and he can do no better than that at node (b). It is still possible, however, that the max^n value at (b) can affect the final max^n value of the tree.

For instance, if the actual max^n value of (b) is (4, 0, 6), Player 2 will prefer the move (6, 4, 0) at node (a), and this will be the max^n value of the game tree. But, if the max^n value of (b) is (0, 4, 6), Player 2 will prefer this value, and so Player 1 will choose (5, 4, 1) to be the max^n value of the game tree. Thus, in general deep pruning is not valid.

4 New Pruning Techniques

We now present our new algorithms. They have the same minimum requirements to prune as shallow pruning, namely a lower bound on each player's score and a upper bound on the sum of scores. This is slightly weaker than requiring a game to be constant-sum, meaning that the sum of all scores at every node is constant, but in either case we can usually make adjustments to the evaluation function in a game to be able to prune effectively.

4.1 Requirements for Pruning

Last-branch and speculative pruning both examine portions of max^n game trees to find nodes whose max^n value can never be the max^n value of the tree. They take different approaches, however, when they prune these nodes.

Returning to Figure 3, we know that Player 1 will never get a better value than 5 at node (b). But, to prune at (b) correctly, we must show that Player 1 cannot get a better max^n value at the root from either node (b) or node (a), as the values at (a) may interact with unseen values at (b) to affect Player 2's, and thus Player 1's move. In this case, deep pruning failed because the value at the right child of (a) was better for Player 2 than a previous child of (a). If the children of (a) were ordered optimally for Player 2, or if there was no right child at (a), the deep prune could not have affected the max^n value of the tree.

While shallow pruning only considers 2 players' bounds when pruning, we can actually use n players' bounds in a n-player game. Since each player has already searched one or more branches when we reach node (b) in Figure 4, we have a lower bound on each player's score. In this case, Player 1 has a lower bound of 5 from the left branch of the root, Player 2 has a bound of 3 from the left branch of (a), and Player 3 has a bound of 2 from the left branch of (b). The sum of these bounds is 10, which is greater than or equal to *maxsum*. We can thus show that any unseen value at (b) cannot be the max^n value of the tree.

Lemma 1: If, in a max^n game tree, the sum of lower bounds for a consecutive sequence of unique players meets or exceeds *maxsum*, the max^n value of any child of the last player in the sequence cannot be the max^n value of the game tree.

Proof: We provide a proof by contradiction. Figure 5 shows a

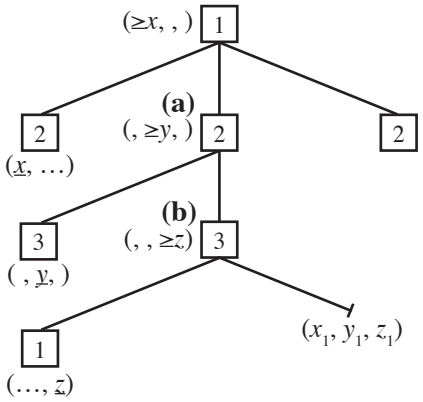

Figure 5. Combining scores to limit value propagation in general.

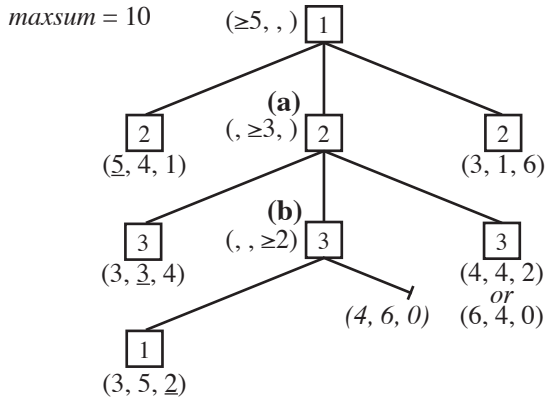

Figure 6. Speculative pruning a max^n game tree.

generic 3-player game tree. In this figure Player 1 has a lower bound of x at the root, Player 2 has a lower bound of y at (a), and Player 3 has a lower bound of z at (b). Given that these values sum to *maxsum*, assume there is a value v at the right child of (b) which will be the max^n value of the game tree.

Let $v = (x_1, y_1, z_1)$. For v to become the max^n value of the tree, each player must prefer this move to their current move. Since ties are broken to the left, z_1 must be strictly better than z, y_1 must be strictly better than y, and x_1 must be strictly better than x. Thus, $z_1 > z$, $y_1 > y$ and $x_1 > x$. So, $x_1 + y_1 + z_1 > x + y + z \geq maxsum$, and $x_1 + y_1 + z_1 > maxsum$. But, by the definition of *maxsum*, this is impossible. So, no value at the right child of (b) can be the max^n value of the game tree. While this is the 3-player case, it clearly generalizes for n players. □

While we have shown that we can combine n players' scores to prove a max^n value will not propagate up a max^n tree, we must also show that a prune in this case will not affect the max^n value of the entire game tree. Last-branch and speculative pruning address this problem in similar ways. Neither algorithm, however, will prune more than n levels away from where the first bound originates.

4.2 Last-Branch Pruning

When a sequence of players have bounds appropriate for pruning under lemma 1, last-branch pruning guarantees that the prune will be correct by only pruning when the intermediate players in the sequence are searching their last branch.

We can see this in Figure 4. To prune correctly, we observe that after searching the left children of node (a) Player 2 has only two choices: the best max^n value from a previous branch of (a), or the max^n value from (b). If Player 2 chooses the max^n value from the previous branch of (a), (3, 3, 4), Player 1 will get a lower score at (a) than his current bound at the root. Lemma 1 shows that the max^n value at (b) can be better than the current bound for Player 2 or for Player 1, but not for both players. So, if Player 2 chooses a value from (b), it must also have a lower max^n value for Player 1 than his bound at the root. Thus, Player 1 will not get a better score at (a), and we can prune the children of node (b).

For last-branch pruning to be correct, in addition to the conditions from lemma 1, Player 2 must be on his last branch, and the partial max^n value from Player 2's previously searched children must not be better for Player 1 than his current bound at the root. In the n-player case, all intermediate players between the first and last player must be searching their last branches, while Players 1 and n can be on any branch after their first one.

```
specmax^n(Node, ParentScore, GrandparentScore)
{
    best = NIL; specPrunedQ = NIL;
    if terminal(node)
        return static eval(node);

    for each child(Node)
        if (best[previous Player] <= ParentScore)†
            result = specmax^n(next child(Node),
                        best[current Player], ParentScore);
        else
            result = specmax^n(next child(Node),
                                    best[current Player], 0);
        if (best == NIL)
            best = result;
        else if (result == NIL)
            add Child to specPrunedQ;
        else if (best[current Player] < result[current Player])
            best = result;
            if (best[previous Player] > ParentScore)
                re-add specPrunedQ to child list;
        if (GrandparentScore+ParentScore+
                                best[current Player] > maxsum)
            return NIL;
    return best;
}
```

Figure 7: Speculative max^n pseudo-code for a 3-player game.

Last branch pruning has the potential to be very effective. Instead of only considering 2 players' scores, it compares n players' scores. In fact, when all nodes in the tree have the exact same evaluation, last-branch pruning will always be able to prune, while shallow pruning will never be able to.

The only drawback to last-branch pruning is that it only prunes when intermediate players between the first and last player are all on the last branch of their search. For a game with branching factor 2 this is already the case, but otherwise we use speculative pruning to address this issue.

4.3 Speculative Max^n Pruning

Speculative pruning is identical to last-branch pruning, except that it doesn't wait until intermediate players are on their last branch. Instead, it prunes speculatively, re-searching if needed.

We demonstrate this in Figure 6. At the root of the tree, Player 1 is guaranteed 5 points. At node (a), Player 2 is guaranteed 3, and at node (b), Player 3 is guaranteed 2. Because 5 + 3 + 2 ≥ *maxsum*, we could prune the remaining children of (b) if node (b) was the last child of node (a).

Suppose we do prune, and then come to the final child of node (a). If the final child of node (a) has value (4, 4, 2), we know Player 1 will not move towards (a), because no value there can be better for Player 1. But, if the value at (a) ends up being (6, 4, 0), the partially computed max^n value of (a) will be (6, 4, 0). With this max^n value, Player 1 will choose to move towards node (b). Because this has the potential to

b	$b^{2/3}$	asymptotic b
2	1.5874	1.8393
3	2.0801	2.4675
4	2.5198	3.0000
5	2.9240	3.4755
10	4.6416	5.4191
1000	100.00	103.61

Table 8: Branching factor gains by speculative max^n in a 3-player game.

change the max^n value at the root of the tree, we will have to search node (b) again using new bounds. This occurs when Player 2's nodes are ordered suboptimally. With an optimal node ordering we will never have to re-search a subtree.

In general, we have to re-search pruned branches if, on a mid-level branch, we find a new value for which both that mid-level player and his parent have better scores. As with last-branch pruning, we can only prune when Player 1 prefers his current move over Player 2's partial max^n value. If we wish to preserve the order of tie-breaking in the tree, we must also retain some extra information about the order of nodes expanded. Nodes that can be pruned by last-branch pruning will be always be pruned by speculative pruning. Pseudo-code for speculative pruning can be found in Figure 7.

As can been seen, it is reasonably simple to perform speculative pruning in practice. In the 3-player implementation, the specmax^n function takes 3 arguments, the current node, the best score at the parent node, and the best score at the grandparent node.

At the line marked † we are checking to see if our parent can get a better score from the partial max^n value of this node than from one of his previously searched nodes. If this is the case, we cannot use the parent's bounds to help prune the children of the current node.

When a node is speculatively pruned, the specmax^n function returns NIL. All nodes that have speculatively pruned children are added to a list of pruned nodes, and if a child is found with a max^n value that better for both the current node and the parent node, then the speculatively pruned nodes will have to be re-searched. This pseudo-code assumes that players always alternate plays in the tree, as in Chinese Checkers. In card games, where this may not be the case, additional checks are needed.

For 3 players, the best-case analysis of speculative pruning can be formulated as a recurrence, the solution of which is the equation $x^3 - x^2 - (b\text{-}1)\cdot x - (b\text{-}1)^2 = 0$. A complete derivation of the recurrence can be found in [Sturtevant, 2003]. Solving this equation for x gives the asymptotic branching factor, which, as b grows large, is $b^{2/3}$. For a general n-player game, the equation will be $x^n - x^{n-1} - (b\text{-}1)^1\cdot x^{n-2} - (b\text{-}1)^2\cdot x^{n-3} - \ldots (b\text{-}1)^{n-1}$. As b grows large, this approaches $b^{n-1/n}$. We give sample values for b, given an optimal ordering of nodes in a 3-player game, in Table 8. The first column contains sample values for b, the second column contains $b^{2/3}$, and the third column contains the actual optimal asymptotic value of b as calculated from these equations.

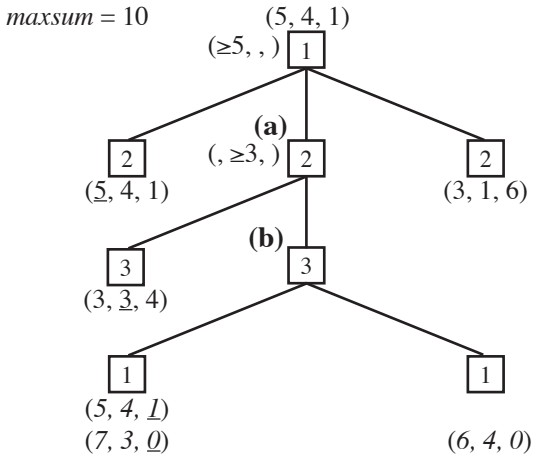

Figure 9. Discrete cut-off evaluations

4.4 Discrete Cutoff Evaluations

It is possible to use tighter bounds for pruning when the evaluation function has discrete as opposed to continuous values. This draws from the proof of lemma 1. In this proof we see that, for a value to affect the max^n value of the tree, $x_1 > x$, $y_1 > y$, and $z_1 > z$. Suppose the minimum delta of a player's score is μ. Since all players in the game must do strictly better than their previous value we can combine this into our bounds.

We demonstrate this in Figure 9. At the root of the tree, Player 1 is guaranteed a score of 5, and at node (a) Player 2 is guaranteed 3 points. In this example $\mu = 1$, so for these players both to prefer to move towards (b) they must get at least 6 and 4 points respectively. Because *maxsum* is 10, we know if Player 3 gets more than 0 points, Players 1 and 2 can't both get better than their current best scores. So, instead of pruning when Player 3 gets 10 - 5 - 3 = 2 points, we can prune when Player 3 gets 1 point.

We can then use our tie-breaking rule to improve this. Because ties are broken to the left, we can prune if Player 3 gets 0 points at the left branch of (b) and Player 1 and 2 don't get 6 and 4 points respectively. If, for instance, the score is (7, 3, 0), Player 2 won't choose this value over the left branch of (a). In addition, Player 3 will only choose a better value than 0 from the unexpanded children of (b), which will meet our earlier conditions for pruning.

It follows from this that we can always prune if $\sum scores \geq maxsum - \mu\cdot(n - 2)$, where $\sum scores$ are the current bounds for the players in the tree. Additionally, we can prune if $\sum scores \geq maxsum - \mu\cdot(n - 1)$ and if on the first branch of the node being pruned the other $n - 1$ players don't all have better scores than their current best bound.

5 Experimental Results

As speculative pruning includes last-branch pruning as a special case, we only report our experiments with speculative max^n. In each experiment, re-expansions by speculative max^n are counted as part of the total number of node expansions, and node re-expansions in speculative pruning never out-

Chinese Checkers expansions at depth 6	
Plain Max^n	1.2 million
Speculative Max^n	100k

Table 10: Average expansions by max^n in Chinese Checkers.

weighed the nodes savings from the additional pruning.

Also, while our sample search trees show cutoff evaluation functions that are seemingly independent of the other players, the actual cutoff evaluation functions used in these experiments are based on the scores of all players in the game. So, instead of just trying to maximize their own score, players are actually trying to maximize the difference between their score and their opponent's scores.

Finally, all our experiments on card games were run on the 3-player perfect-information variations of each game.

5.1 Chinese Checkers

In the past, max^n search of Chinese Checkers trees has been limited to brute-force search, as the bounds in the game are not appropriate for shallow or branch-and-bound pruning. In our experiments we ordered moves according to which ones move a piece farthest across the board. Because the branching factor is very high (over 100 in the mid-game), we only considered the top 10 moves in our ordering at each node.

We played speculative max^n in thirty 3-player games; each game is about 50 moves long. We then measured the average number of nodes expanded at depth 6 by speculative max^n. Results are found in Table 10. While it takes 1.2 million nodes to complete iterative searches to this depth with no pruning, speculative max^n expands an average of 100k nodes, while an optimal ordering of nodes would expand 84,927 nodes for this branching factor and depth. In some cases this optimal ordering is achieved in practice.

5.2 Hearts

It is more difficult to measure the efficiency of speculative max^n in card games. This is because node expansions are highly dependant on the cards in your hand. Additionally, better play, resulting from deeper search, often results in more nodes being expanded throughout a game, as good plays early in the game force more computation later in the game. So, we measured our results in a slightly different manner in Hearts.

We played 300 hands of Hearts with a 500k node search limit. Hands were searched iteratively deeper until the node limit was reached, and re-expansions in speculative max^n counted towards the node limit. We then measured the average search depth by max^n and speculative max^n over all games. The results are found in Table 11. Averaging the search depth over all moves is misleading, because at many points in a hand the search depth is limited by the number of cards in your hand. Instead, we averaged the search depth for points in the tree where the search wasn't limited by the number of cards in the hand, and we also found the average point at which a hand could be searched to completion.

Speculative max^n can search a hand to completion when, on average, there are 22.1 cards remaining in a hand, while max^n can only do it when 20.9 cards remained. In the case where search isn't depth limited, speculative max^n can search an average of 12.1 moves ahead, while max^n can only search 11.1 moves deep.

There are three reasons we don't see more spectacular depth gains. First, in card games the order of player moves in the tree is less uniform, which will lessen the amount of pruning available. Second, we know that the ordering heuristic we used in Hearts can be improved, as it had to re-search many more times than in Chinese Checkers and Spades. Finally, the asymptotic branching factor of Hearts is much lower than Chinese Checkers, so the actual gains are smaller.

5.3 Spades

In Spades, like Hearts, we played 300 games with a 500k node search limit, and then measured the average search depth over all games. The results are also found in Table 11. While speculative max^n could search a hand to completion when it contained, on average, 24.3 cards, max^n could only do it when 21.7 cards remained. When the search was not depth limited, speculative max^n was only able to search depths 15.4 while max^n could only search to depth 11.1.

5.4 Performance against Paranoid Algorithm

The paranoid algorithm [Sturtevant and Korf, 2000] is another algorithm for playing multi-player games. It reduces a game to a 2-player game by assuming one's opponents have formed a coalition. This represents a different decision rule than standard max^n. The trade-off for this less plausible decision rule is a gain in search depth. All techniques from 2-player game research can be used under the paranoid algorithm, and alpha-beta pruning in a n-player paranoid game tree will, in the best case, reduce the number of nodes expanded in a game tree with n players from b^d to $b^{d \cdot (n-1)/n}$.

Although paranoid and speculative max^n have the same asymptotic growth, paranoid will produce smaller trees in practice because it can prune more than n levels away from

	Hearts average search depth		Spades average search depth	
	complete tree	partial tree	complete tree	partial tree
Plain Max^n	20.9	11.1	21.7	11.1
Speculative Max^n	22.1	12.1	24.3	15.4
Paranoid	25.8	17.1	33.5	22.4

Table 11: Average search depth with 500k node expansions in Hearts and Spades.

	Hearts	Spades
Plain Max^n / Paranoid	8.50 / 8.50	5.55 / 5.78
Spec. Max^n / Paranoid	8.01 / 8.99	5.72 / 5.61

Table 12: Average score in Spades and Hearts.

where a bound originates.

A comparison between paranoid and max^n can be found in [Sturtevant, 2002]. Given the gains from speculative max^n, it is worth making a comparison against paranoid.

It has been shown experimentally [Sturtevant, 2002] that the paranoid algorithm outperforms standard max^n at fixed depths in Chinese Checkers. Since adding speculative pruning doesn't change max^n's decision rule, we don't see a gain in max^n's performance against paranoid with speculative max^n. But, it may now be worth considering strategies that are more mixed between standard max^n and the paranoid strategy, as paranoid's advantage in search depth has been greatly lessened.

To measure the differences between paranoid and max^n in card games we followed the same strategy in Hearts and Spades. In both of these games we played 50 hands with a 500k node search limit. Each of these hands were played multiple times to account for player positions on the table. We measured the average search depth for paranoid over these games, and we also compared the average score for max^n with and without speculative pruning. Both algorithms expand nodes at the same rate, so there is no significant difference in CPU time between the two algorithms.

In Table 11 we see that paranoid was able to search deeper than speculative max^n in both Hearts and Spades. In Spades, paranoid can do highly efficient zero-window searches [Pearl, 1980], allowing it to search entire hands when they contain 33.5 cards on average, over 9 cards better than speculative max^n. In Hearts, paranoid can search 3.7 cards deeper than speculative max^n.

But, despite paranoid's advantage in search depth, these results do not translate into quality of play. We see this in Table 12. Each entry in the table contains the listed algorithm's average score followed by paranoid's average score. In Hearts, paranoid and max^n had virtually the same score after all games. But, with the additive pruning and search depth in speculative max^n, it was able to beat paranoid by almost 1 point per hand. (Lower scores are better.) In Spades, where max^n did slightly worse than paranoid, speculative max^n was able to do better than paranoid, despite a much shallower search. (Higher scores are better.) These results show that given speculative pruning, max^n has the potential to be the best decision rule for card games.

6 Conclusions and Future Work

We have seen here that last-branch and speculative pruning combine to provide large theoretical and practical gains for searching max^n trees. This is the first pruning technique to be developed for max^n that is effective on a wide range of games. While other pruning techniques have required very specific cutoff evaluation functions in order to prune, last-branch and speculative max^n pruning only require a game to have a lower bound on each player's score and an upper bound on the sum of all scores.

Also, while the paranoid algorithm must make assumptions about the strategies its opponents use in order to prune, last-branch and speculative pruning make no such assumptions, but can still be adapted to account for opponents strategies. Thus, these algorithms are quite promising for use in multi-player games.

Speculative max^n pruning only prunes over n levels in the game tree. It is possible to extend this idea to prune even deeper into the tree. However, it is unclear if the cost of bookkeeping and re-search for these cases offsets the potential gains. This is an area of our ongoing research.

Acknowledgments

We would like to thank Rich Korf and Alex Fukunaga for their comments and suggestions on these techniques. This work was supported in part by NASA and JPL under contract No. 1229784 and by NSF contract EIA-0113313.

References

[Ginsberg, 2001] GIB: Imperfect Information in a Computationally Challenging Game, Journal of Artificial Intelligence Research, 14, 2001, 303-358.

[Knuth and Moore, 1975] An Analysis of Alpha-Beta Pruning, Artificial Intelligence, vol. 6 no. 4, 1975, 293-326.

[Korf 1991] Multiplayer Alpha-Beta Pruning, Artificial Intelligence, vol. 48 no. 1, 1991, 99-111.

[Luckhardt and Irani, 1986] An algorithmic solution of N-person games, Proceedings AAAI-86, Philadelphia, PA, 158-162.

[Pearl, 1980] Asymptotic properties of minimax trees and game-searching procedures. Artificial Intelligence vol 14 no 2, 1980, 113-138.

[Pearl, 1984] *Heuristics*, Addison-Wesley, Reading, MA, 1984.

[Sturtevant, 2002] A Comparison of Algorithms for Multi-Player Games, *Proceedings of the 3rd International Conference on Computers and Games*, 2002.

[Sturtevant and Korf, 2000] On Pruning Techniques for Multi-Player Games, Proceedings AAAI-00, Austin, TX, 201-207.

[Sturtevant, 2003] Multi-Player Games: Algorithms and Approaches, PhD Thesis, UCLA.