Current Challenges in Multi-Player Game Search

Nathan Sturtevant
University of Alberta, Computer Science Department
Edmonton, AB T6G 2H6
nathanst@cs.ualberta.ca

Abstract. Years of work have gone into algorithms and optimizations for two-player perfect-information games such as Chess and Checkers. It is only more recently that serious research has gone into games with imperfect information, such as Bridge, or game with more than two players or teams of players, such as Poker. This work focuses on multi-player game search in the card games Hearts and Spades, providing an overview of past research in multi-player game search and then presents new research results regarding the optimality of current search techniques and the need for good opponent modeling in multi-player game search. We show that we are already achieving near-optimal pruning in the games Hearts and Spades.

1 Introduction

Artificial Intelligence research in the field of two-player games has been quite successful, with well-publicized victories in games such as chess with DEEP BLUE [1] and checkers with CHINOOK [2]. While there are still interesting challenges in these games, there are many other domains with new challenges yet to be explored. Recent work for programs such as POKI in poker [3] and GIB in bridge [4] has started to shift focus to the difficulties associated with imperfect information and opponent modeling.

Another area that has recently received further attention is the area of multi-player game search. Specifically, this is search applied to domains with three or more players or teams of players. While one might expect that most work from two-player games would extend directly to multi-player games, there are many subtleties that have to be considered. This means that the lessons and techniques derived from work in two-player games must be reconsidered for multi-player games. Techniques of prominence from two-player games may no longer be dominant, while good ideas in two-player games that were not effective enough in practice may find their place in multi-player games.

Without question the most prominent technique from two-player games is minimax search with alpha-beta pruning [5], although this is not the only search algorithm used in two-player games. For instance, there have been attempts to do more selective searches using algorithms like MGSS* [6], or attempts to use an opponent model to increase quality of play, such as in M* [7]. But, in general, these techniques have not found widespread usage, as they cannot compete well with the simplicity and search depth available from minimax with alpha-beta. For instance, while M* was able to out-

perform minimax at fixed depth searches, it was unable to do so when the search was limited by node expansions.

The goal of this paper is to provide an overview of the work that has been done in multi-player games and suggest new challenges for future work in the field. The biggest question we wish to address is whether work in multi-player games will follow the route of two-player games, being dominated by algorithms and techniques for deeper search, or whether techniques based on things like selective search or higher-quality shallow search will be more important. As part of that process we present a brief history of multi-player games, new results on the relative size of multi-player game trees in relation to optimally pruned trees, and an overview of some of the techniques and ideas related to opponent modeling that have yet to be explored in multi-player games.

2 Multi-Player Game Search Research History

Work on multi-player games traces back, in some sense, to Nash's original work defining equilibrium points, showing that non-cooperative games have at least mixed equilibriums [8]. More importantly, it was later shown that there is a pure equilibrium for every perfect information finite game [9, 10]. These theorems were first realized in the field of Artificial Intelligence by Luckhardt and Irani in the max^n algorithm [11]. In her PhD thesis, Carol Redfield (Luckhardt) also did work on formalizing the max^n algorithm, coalition formation and pruning [12]. However, pruning was only presented within the context of selectively evaluating the components of each leaf value in the game tree, as opposed to pruning away branches of the game tree, as is done in alpha-beta.

Around the same time David Mutchler showed that max^n displays game-tree pathologies [13], as was originally observed in minimax [14, 15]. That is, searching deeper into the game tree can produce less accurate results. To date, however, no one has shown that these pathologies occur outside of well-structured artificial game trees.

In 1991, Rich Korf published the first thorough analysis of alpha-beta style pruning in max^n [16]. Specifically, without any bounds on the range of leaf values in a max^n game tree, no pruning is generally possible. But, assuming there is an upper bound on the sum of all players' scores, and a lower bound on each player's score, pruning is possible. This pruning can be divided into three classes, immediate pruning, shallow pruning, and deep pruning. Immediate pruning corresponds roughly to a win or loss in a two-player game. Shallow pruning uses the bounds from two consecutive players to prune a game tree. Deep pruning attempts to use the bounds from two non-consecutive players in a game tree, however, is not guaranteed to preserve the max^n value of a game tree, and cannot be applied in general. In the best case, as the branching factor gets large, shallow pruning can reduce the branching factor of a game tree from b to $O(\sqrt{b})$ as b gets large.

No further work in multi-player game search was published until 2000, when Nathan Sturtevant, working with Rich Korf, showed that in some games shallow pruning could never be applied [17]. Specifically, the effectiveness of shallow pruning relies on both the order of nodes in the tree and the range of leaf-values in the tree, as opposed to

alpha-beta, whose effectiveness just relies on node ordering. However, this work also showed that information from monotonic heuristics could be used to prune max^n game trees. What is more, the bounds from shallow pruning can be combined with the bounds from monotonic heuristics to prune even more effectively.

In this same work, the idea of the paranoid algorithm was formally analyzed for the first time. The idea of the paranoid algorithm is to reduce a multi-player game to a two-player game by assuming that all of one's opponents have formed a coalition against you. While this idea had been touched upon in both Mutchler and Luckhardt's work, it had not been seriously considered as a plausible search algorithm, and its properties had not been explored. The paranoid algorithm has the attractive feature that it inherits the ability to use any technique from two-player games, at the cost of using unrealistic opponent models. In an *n*-player paranoid game tree, alpha-beta pruning will reduce the branching factor from b to $O(b^{n-1/n})$.

The properties of max^n were further explored in [18]. This work brought to light some of the issues with using the max^n algorithm, specifically that tie-breaking is a major issue which limits the applicability of some techniques such as zero-window search [19] from two-player games. It also showed that the paranoid algorithm could be competitive in some games, out-performing max^n in both fixed-depth and node-limited searches in the game of Chinese Checkers.

In 2003 there was a significant advance in pruning algorithms for max^n game trees with the introduction of speculative pruning [20]. This is the first pruning algorithm for max^n that, given a constant-sum game, relies only on the ordering of nodes, as opposed to the actual leaf values in the game tree for effective pruning. Asymptotically, it offers the same reduction as the paranoid algorithm, $O(b^{n-1/n})$, however in practice the paranoid algorithm will still dominate speculative pruning.

There is a discrepancy between the best case branching factor of shallow pruning, $O(\sqrt{b})$, and the best case branching factor of paranoid and speculative pruning $O(b^{n-1/n})$. That is, the best case of shallow pruning depends only on the branching factor of a game, not on the number of players in the game. This discrepancy is explained in more detail in [21], but arises from the assumption of independent sub-trees in a max^n tree. In practice, the best case for shallow pruning actually converges to best-case for immediate pruning, where every leave has a value of win/loss. In this case, only a very precise arrangement of leaf values can achieve the best-case, which will be independent of the number of players in the game, but we do not expect anything close to this in practice.

Given this history, we push to answer the question "What techniques should be used to play multi-player games?" The answer to this question for many two-player games is well known: use minimax with alpha-beta, a high-quality heuristic evaluation function, transposition tables, opening and closing books. The answer for multi-player games are still being explored. However, a program developed to play Hearts using speculative pruning, transposition tables, and monte-carlo simulation for imperfect information was able to out-play one of the better commercial Hearts programs [21], although its play is still weak against humans. We explore this question further in the remainder of this paper.

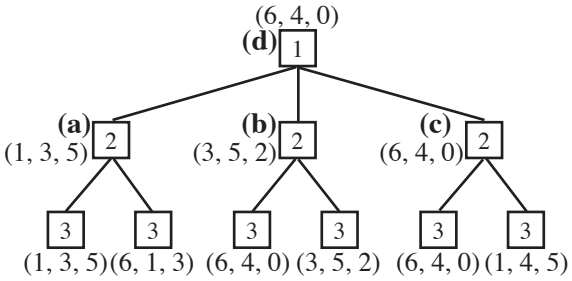

Figure 1: A 3-player max^n game tree

3 Sample Domains

We use two domains, Hearts and Spades, to illustrate the ideas and concepts in this paper. Hearts and Spades are both trick-based card games. In such games cards are dealt out to each player before the game begins. The first player plays (*leads*) a card face-up on the table, and the other players follow in order, playing the same suit as lead 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 face down in his discard pile, and leads the next trick. This continues until all cards have been played.

Hearts is usually played with four players, but there are variations for playing with two or more players. The goal of Hearts is to take as few points as possible. A player takes points when he takes a trick which contains point cards. Each card in the suit of hearts is worth one point, and the queen of spades is worth 13. 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 all 26 points, or "shoots the moon," he instead gets 0 points, and the other players all get 26 points each. These fundamental mechanics of the game are unchanged regardless of the number of players.

Spades can be played with 2-4 players. Before the game begins, each player predicts how many tricks they think they are going to take, and they then get a score based on how many tricks they actually do take. With 4 players, the players opposite each other play as a team, collectively trying to make their bids, while in the 3-player version each player plays for themselves. There are, however, several popular games for which the mechanics of play are identical to Spades, except that players do not play in teams no matter how many players there are. These games have various names and rules that differ by region, but we consider them as larger variations on the game of Spades. More in-depth descriptions of these and other multi-player games can be found in Hoyle et al [22]. A much larger database of games and descriptions can currently also be found online at http://www.pagat.com/.

Although these games are imperfect information games, we will play them as perfect information games, assuming we can see all the cards that our opponents hold.

Given this assumption, we can use monte-carlo methods in real play to sample possible distributions of opponent cards. These methods have been applied most successfully in Bridge [4], and to a lesser extent in Hearts [21].

4 The Max^n Algorithm

The max^n algorithm [11] is a general form of the minimax algorithm and can be used to play any *n*-player general-sum game. For two-player games, max^n simply computes the minimax value of a tree. We will generally call what max^n calculates as the max^n value of a game tree, although in a more general sense it is calculating a Nash equilibrium.

In a max^n tree with *n* players, the leaves of the tree are *n*-tuples, where the *i*th element in the tuple is the *i*th 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 *i*th component is maximum. At the leaves of a game tree an exact or heuristic evaluation function can be applied to calculate the *n*-tuples that are backed up in the game tree.

We demonstrate this in Figure 1. In this tree there are three players. At node (a), Player 2 is to move. Player 2 can get a score of 3 by moving to the left, and a score of 1 by moving to the right. So, Player 2 will choose the left branch, and the max^n value of node (a) is (1, 3, 5). Player 2 acts similarly at node (b) selecting the right branch, and at node (c) breaks the tie to the left, selecting the left branch. At node (d), Player 1 chooses the move at node (c), because 6 is greater than the 1 or 3 available at nodes (a) and (b).

4.1 Speculative Pruning

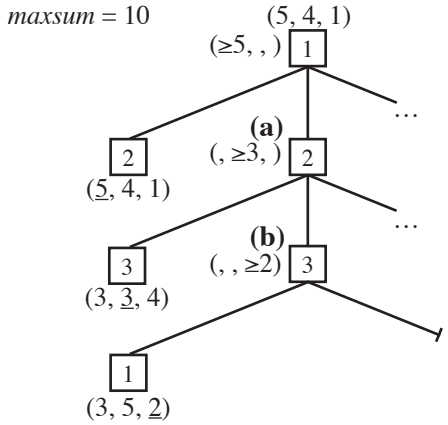

Figure 2. Simple example of speculative pruning in a max^n tree.

Speculative pruning [20] is the most general effective pruning technique for multi-player games. It is a non-directional algorithm, meaning that it may not search through a game tree in a strictly left-to-right fashion. The key idea behind this algorithm is that we prune nodes that may still affect the max^n value of the game tree. But, as the search progresses, we can detect whether the pruned nodes could have actually affected the max^n value of the game tree, and then re-explore portions of the tree as necessary. We demonstrate this in Figure 2, however we leave out some of the details for simplicity.

In this example, the sum of all players' scores, *maxsum,* is 10, and each player's score has a lower bound of zero. At the root of the tree, Player 1 can get at least 5 by moving to his left branch. At node (a), Player 2 can get at least 3 points from his left branch. Finally, at node (b) Player 3 can get at least 2 points from his left branch. The sum of consecutive lower bounds at this point in the game tree, $5 + 3 + 2 = 10$, is at least as large as *maxsum*, so we can prune the right child of (b). The caveat is that unexplored children of (a) may affect the max^n value of the game. But, we can detect this and re-expand node (b) and its children if necessary. It is possible that poor node ordering may cause us to expand more nodes using speculative pruning than we would have without it, although we have never seen this in practice.

4.2 Other Properties of Max^n

We cannot cover all the theoretical details of the max^n algorithm here, but we do need to point out a few issues that strongly affect how the max^n algorithm can be used. A minimax game tree has a single minimax value, no matter what order the game tree is searched. A max^n game tree, however, can have many different max^n values, depending on ties in the game tree and the tie-breaking rule used. This is a property of multi-player games, as each max^n value corresponds to a different equilibrium point in the game. This means that any time we vary node ordering in the game we have the potential of changing the max^n value of the corresponding game tree. Furthermore, the change may not be bounded. For instance, the value at the root of a tree can change from a win to a loss based on the difference of how one tie is broken. How large of a problem this is in practice is not yet fully understood, although we address some of the related issues in the following sections.

5 Comparing Max^n Tree Reductions

In this section we begin our presentation of new research. If we wish to understand how various search enhancements will affect the size of max^n game trees, it is important to have an idea of how current pruning algorithms perform, and whether they are anywhere close to optimal. There are well-known techniques that have been applied to two-player games, such as the history heuristic [23], which have helped to order two-player game trees nearly optimally. If ordering max^n game trees is difficult, we would also like to know if such techniques can help.

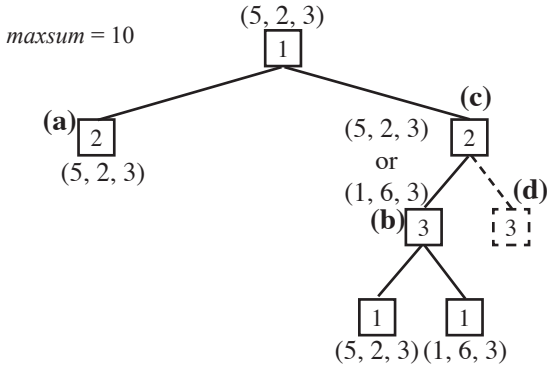

Figure 3: Increasing pruning by modifying tie-breaking rules.

5.1 Estimating Minimal Max^n Trees

Computing the minimal size of a game tree is a non-trivial task, as was originally noted by Knuth and Moore in their analysis of alpha-beta pruning [5]. Specifically, they point out that just ordering nodes from best to worst within a game tree may not be optimal. In an unbalanced tree, for instance, if we are likely to get a pruning cut-off, we might want to search the move that leads to the smallest sub-tree first.

This same issues exist in multi-player games, although there are further complexities. For instance, depending on the ties in a multi-player game, each node in the game tree may have not one, but many different possible max^n values, each one corresponding to a different way of breaking ties in the game tree. Thus, we might try to find the best way to break ties in order to create the smallest game tree possible, although this may lead to a poor max^n value. We might also try to make smaller adjustments that don't change the max^n value of an entire game, but still decrease the portion of the game tree that we must search. We demonstrate this in Figure 3.

In this figure, Player 1 can get a score of 5 at node (a). When searching the right branch of the tree, Player 3 has a tie at node (b). Normally we break ties to the left, which will not affect the max^n value of this game tree. If we do this, however, Player 1's bound at the root, 5, plus Player 2's bound at (c), 2, will not be adequate to prune, as they don't sum to at least *maxsum*, 10. But, breaking the tie at (b) differently to give Player 2 a higher score at node (c) will allow us to prune node (d) and any of its children, since the best scores for Player 1 and Player 2 would then exceed *maxsum*. Thus, by making subtle changes to a max^n game tree, we could increase pruning. We haven't attempted this in practice.

When we prune a *n*-player max^n game tree, every tree fragment where pruning occurs has a similar structure to Figure 2, meaning we do not usually prune more than *n*-1 ply away from where any bound originates, unlike in alpha-beta. It is possible to do so correctly, but the bookkeeping costs of making sure such a prune is legal and the potential cost of having to re-search that line under speculative pruning is prohibitive.

Table 1. Tree sizes in Hearts and Spades given a variety of search enhancements.

Game	Full Tree	TT	TT + P	Ordered
Spades	112M	711k	296k	232k
reduction	-	157x	2.4x	1.3x
Hearts	52.7M	2.04M	1.01M	580k
reduction	-	26x	2.0x	1.7x

However, in an optimally ordered tree, such pruning is not only possible, but it will never lead to tree re-expansions, so there will be no additional overhead or bookkeeping costs.

Comparing the effect of such pruning, however, would be misleading, because we would still have to pay such costs in practice. The only time we might consider using such pruning in practice is when we are willing to give up the guarantee that we calculate a correct max^n value, which we will discuss further in section 5.4.

5.2 Current Best Max^n Tree Reductions

Given a best approximation of an optimally ordered game tree to compare against, we can then compare to the best techniques for generating a similar game tree, without omniscient ordering. A good overview of such an attempt in two-player games can be found in [23], which showed that a combination of transposition tables and the history heuristic were able to account for 99% of the possible reductions in a game tree.

We ran four experiments in the games of Hearts and Spades to see how tree sizes compare when using standard techniques. We compared the number of nodes analyzed for the first move in each of the card games. For simplicity, we used a 3-player version of both games, and limited the number of cards in each player's hand so that we could save the game trees to disk. We played a total of 100 hands in each game. Each game tree was saved to disk, with transpositions recorded, to minimize tree size. We then measured the average number of nodes under various combinations of techniques. In Spades, we gave each player 8 cards, for a total search depth of 24 ply, while we gave each player 7 cards in Hearts, for a total search depth of 21 ply. To ensure correctness we used a total ordering on leaf values in the game trees to break ties, which guarantees that each game tree will only have a single max^n value.

Our first experiment was to measure the full size of the game tree, with no search enhancements, besides those inherent in the domain itself. For instance, we only generate one move when a player holds a sequence of consecutive cards in their hand. In Spades these trees were, on average, 112 million nodes, while in Hearts they were 52.7 million nodes. This is the total count of nodes in the tree, not just leaf nodes.

Next, we enabled transposition tables. In Spades, this reduced the size of the game tree to 711k nodes on average, 157x times smaller than the previous tree. In Hearts, transposition tables reduced the average tree size to 2.04 million nodes, a reduction factor of 26x. The reduction is smaller in Hearts because of the significance of the AKQ♠, where in Spades we only care about relative rankings of cards. The next enhancement we added was speculative pruning (which includes shallow pruning). In Spades, this reduced the average tree size to 296k, while in Hearts it reduced it to 1.01 million nodes. Again, this is the total number of nodes in the tree, including re-expansions from speculative pruning.

Finally, we compared the trees with all of these enhancements to ordered trees using transposition tables and speculative pruning. As discussed previously, we cannot measure the true optimal tree sizes, but we can guarantee that the first line of play is the best at every node. Doing so reduced the average Spades tree size to 232k nodes, and the average Hearts tree to 580k nodes.

One enhancement that we did not measure for either of these games is branch and bound pruning. Although it would have reduced the size of these trees further, the reductions are simply based on the properties of the game tree, and are not affected by node ordering.

The most surprising result here is that the game trees are so close to optimal given just a simple manual ordering of nodes. In Spades, for example, this is just composed of a few rules such as "if you can't win the trick, try low cards first" or "if you can use trump to win, try the lowest winning trump first." Also, in Hearts we sometimes observed many re-expansions from speculative max^n, and so we expected these trees to be significantly larger than optimal. But, it appears that transposition tables are able to compensate well for the cost of node re-expansions, although there is more room for improvement in Hearts than in Spades.

Overall these results also explain why, in separate experiments not shown here, we were unable to get significant gains from using the history heuristic, although it was able to perform as well as our custom ordering. In domains where there are not easy or obvious ordering rules, we would still expect it to be effective in improving the efficiency of search.

5.3 Implications

These results have significant implications for future directions of multi-player game research. We are currently within a factor of two times optimal, which is on the order that we often give up with techniques like iterative deepening. As we have achieved near optimal pruning relatively easily in practice, this means that instead of focusing our efforts improving node-ordering, there are two directions to push future research.

The first direction, which we consider in the next section, is to continue to develop pruning techniques and methods to try to extend search depth even further. The second possible direction is to use a shallower search, but to apply more complex reasoning methods to the search in an attempt to improve performance.

Table 2. The effect of different techniques on the calculated max^n value.

	% changed moves	nodes expanded
regular ordering	-	413k
random ordering	58%	618k
approx. deep pruning	10%	120k

5.4 Further Search Techniques

Speculative pruning was the first pruning technique that is generally applicable to multi-player games. We do not expect that there are any other general pruning techniques that are significantly better than speculative pruning in practice, however there are better pruning techniques in theory. For instance, suppose we want to avoid the bookkeeping costs from speculative pruning, and ignore the need to re-expand portions of the game tree. This would only be marginally faster in practice. But, taking this idea further, if we are going to ignore re-expansions we could re-consider deep pruning. That is, expanding the idea of speculative max^n beyond consecutive nodes in the game tree. Normally the bookkeeping and re-expansions costs would be prohibitive, but if we aren't going to re-expand nodes, we might as well get the best gain possible, since we are not guaranteed to preserve the same max^n value in the game tree either way. We call this approximate deep max^n.

This approach may not have much justification in two-player games, but in multi-player games there is some possible justification. First, a max^n game tree may have several max^n values depending on the way ties are broken in the tree. Second, we must model both our opponents leaf utility function and the way they are going to break their ties when they play the game. If we break ties incorrectly, the max^n value we calculate may not be the one our opponents calculate. Thus, since we are prevented in practice from being able to guarantee that our max^n value will coincide with the one our opponents calculate, we can argue that the small amount of noise introduced by deep pruning may be offset by increased search depth. This may be particularly true in card games, where the branching factor decreases the farther we search into the tree. So a small reduction in node expansions may bring a larger increase in search depth.

To investigate this in practice, we generated 100 hands in the three-player variation of the game of Hearts. We then searched these hands to determine which move would be made at the root of the tree using three different methods. The first method was simply to use a predefined node-ordering under speculative pruning. The second method was to use a randomized node ordering under speculative pruning. The third method was to use the predefined ordering with approximate deep pruning. The results of these experiments are in Table 2. Using a random move ordering changed the move that would be made 58% of the time, while using approximate deep pruning changed the move only 10% of the time. As expected, the average number of nodes expanded was higher when we used a random ordering. When using approximate deep pruning we were able to reduce the nodes expanded by over a factor of three. These results sug-

gest that it is possible that any mistakes made from approximate deep pruning may be more than compensated for by the fact that we don't have an exact model of how our opponent will order their moves. Results on quality of play when compared to various opponents, not shown here, were inconsistent, partially due to issues of opponent modeling, which we will discuss in the next section.

We note that there is a similar argument to why we should use the paranoid algorithm. Specifically, because it is a form of minimax, any techniques developed for minimax can be applied to the paranoid algorithm, and in practice the paranoid assumption allows you to search deeper than you can with max^n. But, the paranoid algorithm can also lead to extremely pessimistic play. The one exception to this is in the game of Chinese Checkers. Our conjecture, first suggested in [21], is that if we had some measure *m* of the ability of our opponents to collude, we will only want to use the paranoid algorithm when *m* is low, that is, when it is difficult for them to collude, otherwise we will want to use something more similar to max^n.

6. Opponent Modeling

While there has been a reasonable amount of work on incorporating opponent modeling into two-player games, this work hasn't been widely used in practice. For instance, Jansen [24] describes various positions in a game where you would need to consider your opponents strategy, Iida, et. al. suggested other potential applications of opponent modeling [25, 26], and various other algorithms for opponent modeling have been suggested by Carmel and Markovitch [7], Korf [27], and Donkers et. al [28].

Results from Carmel and Markovitch give a strong indication of why this work hasn't been more widely applied. In their experiments in Checkers they showed that while opponent modeling methods were superior given the same search depth, they weren't effective when budgeted the same number of node expansions. This is because opponent modeling necessarily reduces the amount of pruning possible in a game tree. So, alpha-beta is able to overcome handicaps in opponent modeling due to its advantage in search depth.

Donkers presents further experiments with opponent modeling in [29], showing that in certain situations gains from opponent modeling are possible, although it is a difficult task, even when given a perfect model of one's opponent.

Most work in opponent modeling has been focused on two-player perfect information games. But, in multi-player games we don't see the same large gains from pruning as in two-player games. This means that if there are any benefits to be obtained by good opponent modeling, they are less likely to be overshadowed by deeper search by a version of max^n that does not do sophisticated opponent modeling.

There are several ways we can add opponent modeling to max^n. One way is without modifying the max^n algorithm itself, but instead just by changing the evaluation function used by our opponents at the leaves of the game tree. This is a limited form of opponent modeling that is inherently assuming that the model we have of our opponent is correct, and that our opponent also has a correct model of our own behavior. This approach will not directly allow us to use a sophisticated model of our opponent to trick

them into making sub-optimal plays, because we are assuming that we both have accurate models of each other, but it is a reasonable starting point. We will use this idea in a moment, but we first provide a stronger motivation of why we cannot ignore opponent modeling in multi-player games.

6.1 Opponent Modeling Motivation

A different explanation for why opponent modeling techniques weren't developed earlier and haven't been widely applied in two-player game search is that the assumption of an optimal opponent in the minimax strategy is adequate for most play. While this is true in two-player, zero-sum games, it is not true in multi-player games, because there isn't necessarily a good definition of optimality for our opponents.

For instance, we may define an optimal opponent as one that plays a Nash equilibrium. But, as we have mentioned multiple times, there can be many different Nash equilibriums in a multi-player game, and we do not know which equilibrium our opponent will be playing. Furthermore, if we are each playing parts of different equilibrium strategies, our combined strategy will not necessarily be part of any equilibrium strategy. We might try to avoid this issue by guaranteeing that a game tree only has one equilibrium point, by eliminating all ties in the game tree.

We define a *pure tie* between two $\max^n$ values to be one in which all players have exactly the same scores. Any other tie just assumes that a single player has a tie between their own components of a $\max^n$ value. If every tie in a game tree is *pure*, there will be exactly one $\max^n$ value of the game tree. For many games, it is fairly easy to modify the evaluation function to avoid ties.

But, while this will avoid the equilibrium selection problem, it leaves an equally difficult opponent modeling problem. We now must have a perfect model of our opponent's evaluation function, which is impossible in practice. For instance, one player might be maximizing their own score while another is maximizing the difference between their score and their opponents' score. Thus, the fundamental problem hasn't changed. Unlike two-player games, there is no widely applicable "optimal" opponent model for use in multi-player games.

This can be seen from Korf's proof of why deep pruning fails in multi-player games [16]. In this proof he shows that changing the value of any single node in a $\max^n$ game tree can potentially change the $\max^n$ value of the entire tree. So, if we have modeled our opponent's utility function incorrectly at any node in a game tree, the $\max^n$ value of the tree can change, and our expected utility from our calculated $\max^n$ value may be incorrect. This means, unless we have some oracle providing perfect information about our opponent, we cannot be guaranteed that the $\max^n$ strategy we are playing with is correct. We can now show how this occurs in practice.

6.2 Opponent Modeling Experiments

To understand better the role opponent modeling plays in multi-player games, we ran

Table 3. The six possible ways to assign 'MT'and 'mOT' player types to a 3-player game

	Player 1	Player 2	Player 3
1	MT	MT	mOT
2	MT	mOT	MT
3	MT	mOT	mOT
4	mOT	MT	MT
5	mOT	MT	mOT
6	mOT	mOT	MT

experiments in the games of Hearts and Spades. For each of these games, we chose two different evaluation functions to compare the effects of correct and incorrect opponent modeling. Players were dealt small enough hands (8 cards in Spades, 7 in Hearts) so that they could search entire game trees, meaning the evaluation function is exact.

In Spades, each player bids on how many tricks they expect to take at the beginning of the game, and is trying to take at least that many tricks, with as few overtricks as possible. Taking 10 overtricks leads to a 100 point penalty. The first evaluation function we used for Spades simply tried to maximize the number of tricks taken, so we will call it 'MT'. The second evaluation function tried to both make the bid, but also minimize overtricks, so we will call it 'mOT'. We then created two variations on each of these players. The first variation was aware of the type of player it was playing against, and could model that player's evaluation function as part of its search. When a player is using a model of their opponent, we add a subscript *m* to their player type. If they are not modeling their opponent correctly, each player assumes that all other players are the same type that they are. This gives us four player types, MT, MT_m, mOT, and mOT_m.

Players receive a cumulative score throughout a round until they reach a pre-determined bound, which we set at 200 points, when a winner is declared. For each of the four variations on player types (<MT, mOT>, <MT_m, mOT>, <MT, mOT_m>, <MT_m, mOT_m>) we played 600 rounds and then measured the average score and number of wins. Within these rounds, we repeated each round six times with the same cards being dealt to each position at the table in order to account for all combinations of players and positions in a 3-player game, as shown in Table 3.

The results of this experiment are in Table 4. As expected, when both players had correct models of their opponent, the mOT players outplayed the maximizing players, winning 53% of the games and having 10 more points on average. But, when the mOT player did have not a correct opponent model, it played much more poorly, essentially tying the game when the mOT player was the only player with a correct model, and losing badly when the mOT player did not have a correct model of the opponent.

This is not surprising, since, in Spades in particular, having a bad opponent model can be very costly. This is because of the large penalty for missing your bid. If you are also trying to minimize overtricks, you must be certain that you will first get the chance

Table 4: Spades Games.

	"maximizing tricks" player		"minimize overtricks" player	
comparison	average score	% wins	average score	% wins
MT_m v. mOT_m	139.0	47.0	149.2	53.0
MT_m v. mOT	145.0	65.0	78.0	35.0
MT v. mOT_m	144.9	49.3	144.1	50.7
MT v. mOT	147.9	63.2	95.8	36.8

to make your bid. An incorrect assumption of how your opponent is going to play can break this assumption and carry a high penalty.

We performed similar experiments in the game of Hearts. In Hearts we played until one player had a score that exceeded 100, where players are trying to minimize their scores. When a round ends, the player with the lowest score wins. Our first player type simply tried to minimize their score (mS), while the second player tried to maximize their lead (ML) over the next best player of they were ahead, and tried to minimize the amount they trailed the best player if they were behind. The results are in Table 5. We see that the players trying to minimize their score were generally successful at that task, but not as successful in the larger goal of actually winning games. When both players modeled each other correctly, the ML player averaged 92 points per round versus 86.6 for the minimizing player, but won 54.5% of the total games. When the ML player had a model of the minimizing player, but not vice versa, the ML player won 58.1% of the games with a slightly lower average score of 89.3. On the other hand, when the ML player didn't have a proper model of the opponent, and the mS player did, the ML player only won 45% of the games, and averaged 95.1 per game. Finally, when neither player had a model of each other, they both won roughly the same number of games, although the ML player still had a much higher (worse) average score.

In both of these games, our experiments show that having a good opponent model is important, but if the modeling done is incorrectly, it can have strongly negative repercussions. We note that except for the case in which both players had models of each other, the modeling is incomplete. Specifically, when a player had a model of their opponent, they also assumed that their opponent had a model of them, which wasn't always true.

Table 5: Hearts Games

	"minimizing score" player		"maximizing lead" player	
comparison	average score	% wins	average score	% wins
mS_m v. ML_m	86.6	45.5	92.0	54.5
mS v. ML_m	86.5	41.9	89.3	58.1
mS_m v. ML	72.7	54.5	94.4	46.5
mS v. ML	74.4	49.1	93.2	50.9

In order to correct this deficiency in modeling, we need to extend M* or a similar algorithm to *n*-player games, as max^n cannot do full opponent modeling by only changing the leaf evaluations in the game tree.

7. Conclusions

In this work we have shown that existing pruning and tree reduction techniques are adequate to produce multi-player game trees to within a factor of two of optimal sized trees. This, along with results on opponent modeling, suggest that a key question for multi-player game tree search is whether we better off trying to search as deep as possible into a game tree, or is it be better to spend our resources doing a more intelligent, but shallower search.

We have taken the first steps towards answering this question by suggesting that using an approximate search strategy like approximate deep max^n will allow us to search deeper into a game tree, and the penalty for doing an approximate search may be no more than what we pay for having an incorrect opponent model. We have also demonstrated how opponent models are crucial to the quality of play in a multi-player search algorithm.

There are many areas in which this work can be extended. First, we could consider extending other two-player algorithms besides minimax to multi-player games. Given this, we also need to compare the relationship between quality of play to search depth, as opponent modeling necessarily reduces search depth. Additionally, there is the question of where we can get an opponent model in practice.

In conclusion, there are many open questions in the field of multi-player game-tree search. Experimental and theoretical results in this paper point to the fact that we may not be able to get by with just brute-force search as we often have in two-player games, although further research must be done to give more conclusive results.

Acknowledgments

We wish to thank the reviewers and Akihiro Kishimoto for their comments and suggestions after reading this paper. This research has also benefited from conversations with Darse Billings, Michael Bowling, Martin Müller, and Jonathan Schaeffer.

References

1. Hu, F.: Behind Deep Blue. Princeton University Press (2002)
2. Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P., Szafron, D. A world championship caliber checkers program, Artificial Intelligence, **53** (1992) 273-290.
3. Billings, D., Peña, L., Schaeffer, J., Szafron, D., Using Probabilistic Knowledge and Simulation to Play Poker, AAAI-99, 697-703, 1999.
4. Ginsberg, M., GIB: Imperfect Information in a Computationally Challenging Game, Journal of Artificial Intelligence Research, **14** (2001) 303-358

5. Knuth, D., and Moore, R., An Analysis of Alpha-Beta Pruning, Artificial Intelligence, vol. 6 no. 4, 1975, 293-326.
6. Russell, S.J. and Wefald, E.H., On optimal game-tree search using rational meta-reasoning. IJCAI-89 (1989), Detroit, MI, 334-340
7. Carmel, D. and Markovitch, S., Incorporating Opponent Models into Adversary Search, AAAI-96, Portland, OR (1996)
8. Nash, J. F., Non-cooperative games, Annals of Mathematics, 54, 286-295, 1951.
9. Jones, A.J., *Game Theory: Mathematical Models of Conflict*, Ellis Horwood, West Sussex, England, 1980.
10. Kuhn, H.W., "Extensive Games and the Problem of Information,", *Contributions to the Theory of Games*, editors Kuhn and Tucker, Princeton University Press, New Jersey, 1953.
11. Luckhardt, C. Irani, K., An algorithmic solution of N-person games, Proceedings AAAI-86, Philadelphia, PA, 158-162.
12. Luckhardt, C., N-Person Game Playing and Artificial Intelligence, PhD. Thesis, University of Michigan,1989.
13. Mutchler, D., The Multi-Player Version of Minimax Displays Game-Tree Pathology, Artificial Intelligence, **64**, (1993), 323-336.
14. Nao, D.S., Pathology on game trees: A summary of results, Proceedings AAAI-80, Stanford, CA, 102-104, 1980.
15. Beal, D.F., An analysis of minimax, In Clarke, M.R.B., editor, *Advances in Computer Chess 2*, Edinburgh University Press, Edinburgh, Scotland, 103-109, 1980.
16. Korf, R., Multiplayer Alpha-Beta Pruning. Artificial Intelligence, vol. 48 no. 1, 1991, 99-111.
17. Sturtevant, N., and Korf, R., On Pruning Techniques for Multi-Player Games, Proceedings AAAI-00, Austin, TX, 201-207.
18. Sturtevant, N., A Comparison of Algorithms for Multi-Player Games, Proceedings of the 3rd International Conference on Computers and Games, 2002.
19. Pearl, J., *Heuristics*, Addison-Wesley, Reading, MA (1984)
20. Sturtevant, N., Last-Branch and Speculative Pruning Algorithms for Max^n, Proceedings IJCAI-2003, Acapulco, Mexico.
21. Sturtevant, N., Multi-Player Games: Algorithms and Approaches, PhD Thesis, (2003) UCLA.
22. 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.
23. Schaeffer, J., The History Heuristic and Alpha-Beta Enhancements in Practice, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 11, No 11, (1989), 1203-1212
24. Jansen, P., Problematic Positions and Speculative Play, In Marsland, t., and Schaeffer, J., eds., *Computers, Chess and Cognition,* Springer New York (1990) 169-182.
25. Iida, H., Uiterwijk, J., van den Heric, H, Herschberg, I., Potential Applications of Opponent-Model Search, Part 1: The domain of applicability, ICCA Journal, 16(4), (1993) 201-208
26. Iida, H., Uiterwijk, J., van den Heric, H, Herschberg, I., Potential Applications of Opponent-Model Search, Part 2: Risks and Strategies, ICCA Journal, 17(1) (1994) 10-14
27. Korf, R., Generalized Game Trees, IJCAI-89 (1989) 328-333
28. Donkers, H.H.L.M., Uiterwijk, J.W.H.M., and Herik, H.J. van den, Probabilistic Opponent-Model Search, Information Sciences, Vol. 135, Nos. 3-4, (2001) pp. 123-149.
29. Donkers, H.H.L.M., Nosce Hostem: Searching with Opponent Models, PhD Thesis, 2003, University of Maastricht.