Robust Game Play Against Unknown Opponents

Nathan Sturtevant
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
nathanst@cs.ualberta.ca

Michael Bowling
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
bowling@cs.ualberta.ca

ABSTRACT

A standard assumption of search in two-player games is that the opponent has the same evaluation function or utility for possible game outcomes. While some work has been done to try to better exploit weak opponents, it has only been a minor component of high-performance game playing programs such as Chinook or Deep Blue. However, we demonstrate that in games with more than two players, opponent modeling is a necessary component for ensuring high-quality play against unknown opponents. Thus, we propose a new algorithm, soft-max^{n}, which can help accommodate differences in opponent styles. Finally, we show an inference mechanism that can be used with soft-max^{n} to infer the playing style of our opponents.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems—*Games*

General Terms

Algorithms

Keywords

multi-player games, search, opponent modeling

1. INTRODUCTION AND BACKGROUND

The minimax decision rule is well-known for its performance guarantees in perfect-information game trees. If we have a perfect evaluation function, minimax gives a lower-bound guarantee on our score, regardless of our opponent. This is different than being maximally exploitive, however, because minimax will not necessarily take maximal advantage of a weak opponent.

In two-player zero-sum games, different algorithms have been suggested with the intent of using opponent modeling to take better advantage of opponent weaknesses [2, 3, 7]. While these algorithms work in theory, they require an *a priori* model of the opponent, and the improvement does not offset the loss of deeper search afforded by alpha-beta pruning in two-player games.

In games with more than two players or teams of players we face a different situation. Game theory does not provide tight lower-bounds on performance in multi-player[1] games as in two-player zero-sum games, and existing algorithms for play demonstrate great weaknesses when playing new or unmodeled opponents. And, while better pruning techniques have recently been developed [14, 15], they are still not as effective as alpha-beta in two-player games.

This paper examines the role of opponent models for search in large multi-player perfect-information game trees. We make two novel contributions: a more robust algorithm for playing n-player games and a representation and method of inferring opponent preferences in these games.

This paper proceeds as follows: we will first discuss previous algorithms for opponent modeling in more detail, as well as the max^{n} algorithm [10] for playing multi-player games. We will then introduce a sample domain, the card game Spades, and demonstrate that an incorrect opponent model, such as assuming the opponent is using the same evaluation function, can be quite harmful. We introduce a new algorithm, soft-max^{n}, which can better accommodate differences in opponent styles. Finally, given this algorithm, we show how we can infer models of our opponents through observations of their play.

1.1 Related Work

Opponent modeling has been studied in the context of two-player, zero-sum, deterministic, perfect-information games for many years. Some of the earliest work on opponent modeling [8, 6] began with the recursive problem of opponent modeling. If I do not assume that my opponent is identical to me, I must have a model of my opponent. Similarly, if my opponent is also trying to model me, I will also need a model of my opponent's model of me, and so on. At first glance this seems to be a recursive nightmare. But, algorithms like M* [2] are capable of handling a number of opponent models and using them recursively. PrOM [3] expanded on these ideas, but still did not find measurably large success, even given a perfect opponent model to begin with.

In some sense, because most of this work is in the context of two-player, zero-sum games, it is not surprising that it has met with limited success in practice. This is the domain with the strongest theoretical algorithm, minimax, and thus has the least need for opponent modeling. We demonstrate here that in multi-player games there is a need for good opponent modeling. This same observation has been made in the context of poker [1, 13], although the approach in that game has been to model the opponent's strategy

[1]We use the phrase *multi-player* to specifically refer to situations with three or more players, as distinguished from the phrase *two-player*, although most of our statements with regard to multi-player games also apply to two-player non-zero-sum games.

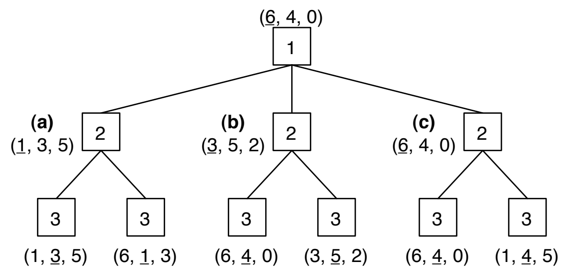

Figure 1: Example max^n tree.

rather than their evaluation function. In many cases the number of outcomes to be ordered is small compared to the domain's state space, making modeling of preferences more tractable.

1.2 Max^n

The max^n algorithm is used for calculating plays in perfect information game trees. 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 or utility for that position. At the interior nodes in the game tree, the max^n value of a node where player i is to move is the max^n value of the child of that node for which the ith component is maximum. Ties are traditionally broken toward the left, to maximize pruning. 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. The player to move is labeled inside each node. 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 the root of the tree, Player 1 chooses the move at node (c), because 6 is greater than the 1 or 3 available at nodes (a) and (b).

1.3 Max^n Theoretical Properties

In order to better motivate the theoretical reasoning for our work, we briefly touch on one aspect of the game theory behind the minimax and max^n algorithms. The minimax algorithm works by computing the best response to any strategy a perfect opponent could play. The value of this strategy is called the minimax value or an equilibrium value. While there may be many strategies that allow a player to achieve this value in a two-player zero-sum game, each of these strategies has the same value.

Max^n calculates equilibrium values and strategies similar to minimax. It is simple to see that max^n returns a line of play which is an equilibrium, because at each point in the game tree players are always taking the move with the maximum score possible. Thus, no player can get a better result by unilaterally changing their strategy. See [10] for a full proof.

In a multi-player game there can be many different equilibrium strategies, but unlike in two-player zero-sum games, they are not guaranteed to all have the same value. Even more importantly, if your opponent is playing a different equilibrium strategy than you are, there is no guarantee that you will be in equilibrium, or that you will even receive the minimum equilibrium value that you expected to receive.[2]

Even in two-player games there are cases where there is a need to distinguish between different strategies with the same equilibrium value. Perhaps the biggest example is from the game of Checkers. In one situation the program CHINOOK was able to prove that a line of play was a draw, when humans mistakenly thought it was a win for the computer. Thus, it appeared to be a bug when CHINOOK made a move that led to a clear draw. In practice CHINOOK just lacked the ability to select between different equilibrium strategies to choose the one that was most likely to lead to an opponent mistake [12].

In the case of Checkers, the result of playing the minimax strategy without attention to an opponent model simply means that the program isn't maximally exploiting a weak opponent. Against a perfect opponent such differences don't matter. However, the concept of a perfect opponent in a multi-player game is ill-defined. In the next section we demonstrate the effects of playing a multi-player game when we have incorrect assumptions or models about our opponents' strategies.

2. SPADES

Most of the work in this paper is not specific to any particular game. But, because the game of Spades is well-known, and it is easy to create small, concrete examples from this game, we will use it as the primary domain for examples and experiments. There are many games similar to Spades which have similar properties, which we will not cover here, including Oh Hell! and Sergeant Major. The complete rules of Spades is found in [4], but many more games are described in detail at http://www.pagat.com/.

Spades is a card game for two or more players. In the 4-player version players play in teams, while in the 3-player version each player is on their own, which is what we focus on. We will only cover a subset of the rules of Spades here. A game is split into many hands. Within each hand the basic unit of play is a trick. At the beginning of a hand, players must bid how many tricks they think they can take in that hand. At the end of each hand they receive a score based on how many tricks they actually took. The goal of the game is to be the first player to reach a pre-determined cumulative score, often 300.

If players make their bid exactly, they receive a score of $10 \times bid$. Any tricks taken over their bid are called overtricks. If a player takes any overtricks they count for 1 point each, but each time a player accumulate 10 overtricks, they lose 100 points. Finally, if a player misses their bid, they lose $10 \times bid$. So, if a player bids 3 and takes 3, they get 30 points. If they bid 3 and take 5, they get 32 points. If they bid 3 and take 2, they get -30 points. Thus, the goal of the game is to make your bid without taking too many overtricks. In this work we consider the perfect-information version of Spades where we can see our opponents cards.

The perfect-information version of Spades is a complex and challenging problem in itself. If we wish to play the imperfect-information version of Spades, Monte-Carlo sampling can be used to turn a perfect-information player into a imperfect-information player, such as was done in Bridge [5].

To demonstrate the effect opponent modeling has on quality of play, we consider two player types, one player who tries to maximize the number of tricks they take in a hand (ignoring the bid), which we call MT. The other player type we consider attempts to minimize the number of overtricks they take, which we call mOT. Then, we varied these players by either giving them a model of

[2]We could get a guaranteed lower bound on our score by reducing the n-player game to a two-player game, assuming all of our opponents are seeking to minimize our score. This approach is overly pessimistic, assuming that not only will opponents arbitrarily sacrifice their own score to reduce ours, but are also entirely coordinated in doing so.

Table 1: The six ways to arrange two player types, A and B, in a three-player game.

	Seat 1	Seat 2	Seat 3
1	A	A	B
2	A	B	A
3	A	B	B
4	B	A	A
5	B	A	B
6	B	B	A

Table 2: The effect of opponent models on play.

	Players A v. B	Player A		Player B	
		Score	%Win	Score	%Win
(a)	mOT$_{MT}$ v. MT$_{mOT}$	248.6	74.7	163.8	25.3
(b)	mOT$_{MT}$ v. MT$_{MT}$	235.4	59.0	199.2	41.0
(c)	mOT$_{mOT}$ v. MT$_{mOT}$	198.2	53.5	191.4	46.5
(d)	mOT$_{mOT}$ v. MT$_{MT}$	178.2	44.0	207.3	56.0
(e)	mOT$_{MT}$ v. mOT$_{mOT}$	212.4	37.2	250.8	62.8
(f)	MT$_{mOT}$ v. MT$_{MT}$	157.3	36.0	218.4	64.0

their opponent, which we designate in the subscript. So MT$_{MT}$ is a player who is attempting to maximize tricks, and assumes the same about all opponents. MT$_{mOT}$ is also trying to maximize tricks, but assumes that all opponents are trying to minimize their overtricks. Players do not do any further recursive opponent modeling.

We arranged the experiments as follows. For each combination of the two player types and opponent models we played 100 games, each of which ended after any player reached 300 points. In each hand players were dealt seven cards from a standard 52 card deck and searched the game trees in their entirety. A rule-based system was used to determine player bids, and all players used the same system. Because there are three players in each game, each game was repeated with the same cards six times to account for different arrangements of players at the table (See Table 2). Although the specific results will vary based on the number of cards, the score limit for ending games, and the rule-system used for bidding, similar trends are found in the data regardless of these parameters.

We report the results of play in Table 2. Each row shows the average score and percentage of wins for the two listed player types averaged over the 600 games played[3] (100 games for each of the six combinations of player types). In each game two of the players will be of the same type and thus that type will have an increased chance of winning the game. This is offset by the fact that in the other half of the games that type will only be one of the three players and so have a lower chance of winning. In summary, if the player types were equal, they would be expected to win half of the games. Similarly, even if a player type wins 100% of the games, its average score would likely be less than 300 because in half of the games one of the losing players must be of the same type as the winner and presumably have a score less than 300.

By the rules of the game, we would expect the player who is minimizing overtricks to do best, and in general they do. This is particularly noticeable when they have a correct model of their opponent as shown in rows (a) and (b).[4] When both players have correct models of each other (a), the mOT player wins 74.7% of the games. If the MT player does not have a correct model of the mOT player (b), the win rate drops to 59.0%. However, if the mOT player also has an incorrect model, the win rate drops to 44.0% and the preferred evaluation strategy can actually lose. This is a high price to pay for an incorrect opponent model. Furthermore, we cannot just assume our opponent is trying to maximize tricks, because we will do quite poorly against another player who is trying to minimize his overtricks as shown in row (e).

[3]All results reported in this paper have 95% confidence of being within 7.2 points or less, which is sufficient to support our claims.

[4]Due to the nature of our experimental setup, one-third of a players' opponents will actually be of their own type. Hence, when we say player A has a correct model of its opponent, such as in row (a), we mean that at least two-thirds of its opponents are correctly modeled. The only situation where a player has a correct model for all of its opponents is player B in rows (e) and (f).

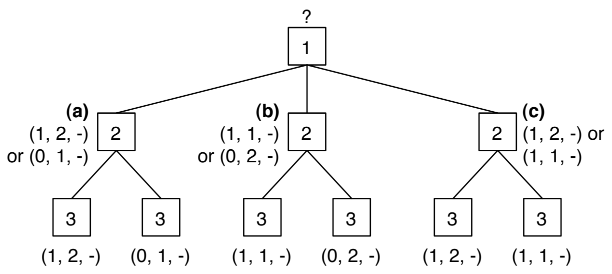

Figure 2: Example search tree in Spades.

Lastly, row (f) shows the same results of the importance of correct modeling when both players try to maximize their tricks. In this case the average scores are much lower than in other games. This occurs because both players are using poor strategies, which increases the variance and thus lowers the average score. Similar results, not reported here, have been seen in other card games.

Note that because we are only doing one level of opponent modeling, it is important that we both have a correct model of our opponent and that our opponent has a correct model of us. Otherwise our predictions of our opponents' behavior will be incorrect. This explains why the results from row (a) are different from row (b). In row (b), the MT_{MT} player will behave differently than the mOT_{MT} player predicted since it expects to be facing an MT opponent. These results indicate that errors in second-level modeling effects (i.e., my model of my opponent's model of me) are important, but of a smaller magnitude than first-level modeling errors.

3. Soft-Max^n

We propose a new algorithm, soft-max^n, which is more robust to unknown opponents. We first present the algorithm intuition in the context of our results in Spades, followed by a more formal presentation of the algorithm.

First let us briefly return to Figure 1. At node (c) there is a tie for Player 2, so either move can be chosen, resulting in a score of 6 or a score of 1 for Player 1. Because max^n only backs a single value up the tree, there is no way for Player 1 at the root to know the risk of taking move (c). Because a value that is a tie for one player is not necessarily a tie for the others, there is a risk any time we encounter a tie in the game tree. The first inclination may be to remove all ties in the game tree. If all players have a publicly known total ordering on possible outcomes in a game (i.e., there are no ties), the max^n algorithm will work "perfectly", in that this problem will never arise. In practice, however, players themselves may only have a partial ordering on outcomes. When there is a tie in the partial ordering, there may be no way to predict how the opponent will break that tie.

Even worse, consider Player 2's decisions in Figure 2, where there are no ties. In this example, both Player 1 and 2 have bid 1

Table 3: Dominance relations in max^n sets.

(a)	(3, 4, 3)
(b)	(2, 1, 7), (4, 2, 4)
(c)	(4, 1, 5), (3, 2, 5)
(d)	(10, 0, 0), (0, 10, 0), (0, 0, 10)

trick, and we will ignore Player 3 for the time being. Consider the possible strategies for Player 2. If Player 2 is trying to maximize tricks, at nodes (a) and (c) the outcome (1, 2, -) is chosen, which allows Player 1 to make their bid, and at node (b) (0, 2, -) is chosen. If Player 2 is trying to minimize overtricks, (1, 1, -) will be chosen at nodes (b) and (c), allowing Player 1 to make their bid, but at node (a), Player 2 will choose (0, 1, -).

If we think Player 2 is trying to maximize tricks, move (a) and (c) will have the same value. Barring additional information in the tie-breaking rules, there is the possibility we will make the wrong move to (a), missing our bid where moving to (c) would have guaranteed our bid. We get similar results if we model Player 2 as trying to minimize overtricks.

It should be clear that there is a tension between two issues. We want to avoid ties in our game trees, because they introduce extra uncertainty in whether we are backing up the correct max^n value. However, avoiding ties requires detailed knowledge of our opponent's preferences, i.e., very specific opponent models. We've already shown, though, that overly specific models can result in poor performance when the opponent's play does not correspond to the chosen model. For example, in Spades it is safe to assume the opponent prefers making their bid to missing it, but it is not safe to assume an opponent will necessarily avoid taking overtricks. Our solution, then, is to use generic, less presumptuous, opponent models, but more intelligently handle the resulting increase in the number of ties. We will first describe the soft-max^n algorithm and its alternate tie-breaking mechanism. In the next section we will describe our formalization of generic opponent models.

3.1 Max^n Sets

Instead of passing a single max^n value up the game tree, the soft-max^n algorithm backs up sets of max^n values. A max^n set s contains max^n values $v_1, \dots, v_k$ where each value v_i is a standard max^n tuple. We combine multiple max^n sets with the *union* operation. The union of two sets is a new set containing all max^n values that appear in each individual set.

We compare sets using a dominance relationship. A max^n set s_1 strictly dominates another max^n set s_2 with respect to some player i if and only if,

$$\forall v_1 \in s_1 \; \forall v_2 \in s_2 \quad v_1[i] > v_2[i].$$

Similarly the max^n set s_1 weakly dominates s_2 relative to Player i if and only if,

$$\begin{aligned} &\forall v_1 \in s_1 \; \forall v_2 \in s_2 && v_1[i] \geq v_2[i] \quad \text{and} \\ &\exists v_1 \in s_1 \; \exists v_2 \in s_2 && v_1[i] > v_2[i]. \end{aligned}$$

We demonstrate these concepts with the values in Table 3. From the perspective of Player 1, (c) weakly dominates (a), because Player 1's scores in (c), 4 and 3, are at least as large as the score of 3 in (a). From the perspective of Player 2, (a) strictly dominates both (b) and (c), while from Player 3's perspective (b) strictly dominates (a), but does not dominate (c). If each player has a minimum score of 0 and a maximum score of 10, the set (d) can neither dominate another set or be strictly dominated by another set.

3.2 The Soft-Max^n Algorithm

Given the definition of a max^n set and dominance relation, we define soft-max^n as played by player i as follows:

1. At a leaf node, the max^n set is a heuristic or exact evaluation function.
2. At an internal node in the tree, the max^n set of values for that node is the union of all sets from its children that are not strictly dominated with respect to player j.
3. At the root of the tree, player i can use any decision rule to select the best set from among the non-dominated moves.

At the root of the tree, there are any number of ways to decide which move to make. For instance, depending on the game state, we may prefer to maximize potential score, minimize risk, or play to receive a guaranteed outcome. For all examples in this paper, we take the move which has the max^n set with the highest average value. There are obvious arguments why this may be the incorrect approach as there's no reason to assume all elements of the set are equally likely, but it is a simple approach that works in practice, as we demonstrate in the next section. We are currently exploring more principled options for making decisions.

We can illustrate soft-max^n using the tree in Figure 2. Recall that we now use only a generic opponent model for each player, which means that we will only distinguish between outcomes where our opponent are choosing to miss or make their bid. Player 1 and 2 each bid one trick. Since Player 2 always takes at least one trick, instead of returning one or the other result at (a), (b), and (c), we return a max^n set containing both child values at these nodes. At the root, Player 1 can, for instance, calculate the worst-case (or average-case) score on each path. Player 1 is guaranteed to take one trick on branch (c), so unlike (a) and (b), this move can be made safely. Thus, under soft-max^n Player 1 can make a more informed decision about which move to make.

The max^n algorithm has been shown to calculate one of many possible equilibria in a game. By comparison, Soft-max^n calculates all equilibria in the game.

LEMMA 1. *Soft-max^n calculates a superset of all pure-strategy equilibria values at every node in a game tree.*

Proof. We provide a proof by induction. First consider the leaves of a game tree. There are only single values in the leaves, so they must be equilibria for those sub-trees. Next, assume that at the kth level in the tree we have a max^n set containing a superset of all equilibria for that subtree. We show by contradiction that at the $(k+1)$th level of the tree we still have a superset of equilibria in our max^n set.

Assume there is a node, n, at the $(k+1)$th level for which soft-max^n does not return a max^n set which contains a superset of all equilibria. This means there is some child c whose max^n set contains an equilibrium not backed up into the parent. But, this can only happen if all values in the max^n set of c are strictly dominated by another child of n. If this is the case, then any individual value in c will always be dominated, and thus no value in c can be part of any equilibrium strategy. This is a contradiction, concluding the proof. □

We can be certain that all equilibria of the game are included in the soft-max^n sets. We cannot guarantee however that additional values will not be included. Informally, the extra values calculated by soft-max^n will be possible outcomes in the game, if we allow players to change their strategies at nodes in the middle of a game where they are indifferent to the possible outcomes.

3.3 Effect on Quality of Play

To demonstrate the effectiveness of soft-max^n, we repeated the experiments reported in Table 2, but this time the player trying to minimize overtricks uses soft-max^n and a generic opponent model. That is, the opponents are only assumed to be trying to make their bid. We employed the same methodology as in the previous experiments, except that now mOT$_g$ refers to the fact that the mOT player has this generic opponent model. The new results are in Table 3.3.

The new "% Gain" column is the increase in win percentage by using a generic model instead of an incorrect model. The generic model results in a substantial increase in the worst-case performance of playing an unknown opponent. In every case except against another mOT player that has modeled it, it is now winning at least half the games. Thus, if we are playing a game where we are unsure of the strategies that our opponents are using, we are better off using a generic model than making the wrong assumption about how our opponents play. The new "% Loss" column shows the further increase in win percentage that could be attained by using a *correct* opponent model rather than the general one. This demonstrates there is still room for improvement if the player can learn a model of its unknown opponent through play. In the next section we present a formalization of opponent models and then show how a specific opponent model can be inferred from opponent decisions.

4. OPPONENT MODELING

In the previous section we showed the improvement gained by using a generic model of the opponent. In this section we formalize a notion of an opponent model and how generic models can be constructed. A simple way to distinguish between two players is based on their evaluation function they use at the leaves of a search tree, that is, their relative utility of the different possible outcomes in the game. If an opponent has a total ordering for the possible outcomes in a game, we can use this information to simulate the decisions they would make in a game. In practice, a player may be ambivalent between two similar states. Thus, we model an opponent with a partial ordering defining their preferences over the possible outcomes in the game.

Formally, an *opponent model*, is a directed graph where the vertices are possible outcomes of the game. Edges are then used to encode the preference relationship. We define an opponent model, O, only by its edge set. In particular, $(u, v) \in O$ (*i.e.*, is an edge in the graph) if and only if u is preferred to v by the player being modeled. The graph thus provides a partial ordering over game outcomes. We assume the graph is closed under transitivity. In other words, if $(u, v) \in O$ and $(v, w) \in O$ then $(u, w) \in O$. Since preferences are necessarily transitive, we can always add edges into any model to form its transitive closure.

Two example opponent models are shown for two-trick Spades in Figure 3. There are six possible ways the two tricks can be taken in the game, which are shown on the left side of the figure. If Player 1 bids one, the models for maximizing tricks and minimizing overtricks are shown on the right. If the first player is trying to maximize tricks, they will prefer outcome 6 where they get two tricks, to all other outcomes. Outcomes 4 and 5 where they get one trick, would be preferred to outcomes 1, 2 and 3, where they get none. If Player 1 instead wants to minimize overtricks, they will have a slightly different model. Player 1 would prefer outcomes 4 and 5 to all other outcomes, and outcome 6 to outcomes 1, 2, and 3. For many domains, like Spades, the number of possible outcomes is relatively small and can be enumerated in a graph. In domains where this is not possible, the graph can be represented and reasoned over implicitly.

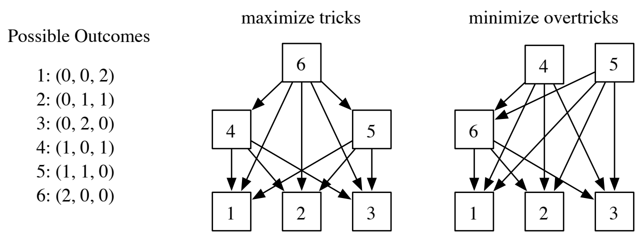

Figure 3: Two example opponent models for a tiny version of Spades.

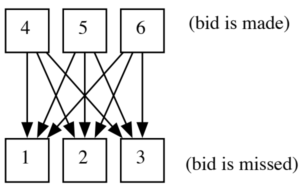

Figure 4: The *generalization* of the models from Figure 3.

It is likely that a player will not have a specific model of his opponent. The player, though, may have a set of candidate models, $O_1, \ldots, O_k$, believing that one of these models accurately captures the opponent's preferences. We saw in Table 2 that an incorrect model can have devastating effects on the quality of play. We also saw that soft-max^n with very generic assumptions about the opponents' preferences can greatly improve play. This suggests that given a set of candidate models we want to construct the *generalization* of these models. This generalization should (i) be as specific as possible, and (ii) be consistent with all of the candidate models. We can achieve this generalization by simple intersection:

$$G(O_1, \ldots, O_k) = \bigcap_{i=1}^{k} O_i.$$

Hence, $(u, v) \in G(O_1, \ldots, O_k)$ if and only if $\forall i\, (u, v) \in O_i$. In other words, u is preferred to v in the generalization if and only if u is preferred to v in all of the candidate models.

In Figure 4 we demonstrate the generalization of the models from Figure 3. In this case, the generalization of the two candidate models results in a model where the opponent prefers outcomes in which they make their bid, but it does not tell us anything beyond that.

4.1 Opponent Modeling and Soft-Max^n

Given an opponent model, it is easy to make use of it in soft-max^n by simply defining a dominance rule on max^n sets. Let s_1 and s_2 be max^n sets. We say s_1 strictly dominates s_2 under opponent model O, if and only if,

$$\forall u_1 \in s_1\ \forall u_2 \in s_2 \quad (u_1, u_2) \in O,$$

and weakly dominates, if and only if,

$$\begin{array}{lll} \forall u_1 \in s_1\ \forall u_2 \in s_2 & (u_2, u_1) \notin O & \text{and} \\ \exists u_1 \in s_1\ \exists u_2 \in s_2 & (u_1, u_2) \in O. & \end{array}$$

These definitions of domination can now be used in the soft-max^n search for any given opponent model. Given a set of candidate opponent models, we can also construct the generalization of these models and use that model with soft-max^n.

Table 4: The effect of generalized opponent models on play.

	Players A v. B	Player A Score	%Win	%Gain	%Loss	Player B Score	%Win
(a)	mOT$_g$ v. MT$_{mOT}$	241.7	68.5	15.0	6.8	178.2	31.5
(b)	mOT$_g$ v. MT$_{MT}$	218.2	53.5	9.5	5.5	206.1	46.5
(c)	mOT$_g$ v. mOT$_{MT}$	242.2	54.8	4.8	8.0	228.7	45.2
(d)	mOT$_g$ v. mOT$_{mOT}$	230.6	46.0	8.8	4.0	243.8	54.0

In domains where we cannot explicitly enumerate all possible outcomes in a game, we can instead use a functional model to provide the same information that we are getting from our explicit models in Spades.

4.2 Inferring Opponent Models

Without a specific opponent model we have shown that effective play can be obtained by generalizing a set of candidate models. During the course of a game, though, we actually observe our opponents' decisions. This opens up the possibility of inferring models of our opponents' preferences from observations of their choices. This is no easy task, as it is not generally possible from observations to distinguish between a player's preference over outcomes and simple indifference. Any decision of the opponent can always be explained as some tie-breaking mechanism on outcomes over which the player has no preference. Inference *ex nihilo* may be ill-defined, but inference based on a set of candidate models can still be done.

Given a set of candidate opponent models and a game with decisions by the opponents, we can identify which models are consistent with their actual decisions, and more importantly which are not. This will not identify which model best captures their preferences, but it can be used to eliminate models which certainly do not capture their preferences.[5] Eliminating these models will make the generalization of the remaining candidates more specific to the actual opponent. For example, near the end of a hand of Spades one may observe the opponent play their final cards so as to avoid taking a second overtrick, which they could have guaranteed taking. From this one can infer that the maximizing tricks model is inconsistent and eliminate it from the candidates. If maximizing tricks and minimizing overtricks were the only two candidates, one would then be certain the opponent minimizes overtricks. In later hands, this more specific model of the opponent can be used.

In order to perform this consistency check we need to reconstruct the actual decisions an opponent faced during the game. We can do this by running soft-max^n with generalized models for all of the players, including ourselves. Hence for any opponent decision we will have max^n sets, which we know contain the possible outcomes the opponent was deciding between. Let s_1 and s_2 be two sets the opponent decided between, where the actual decision was the action associated with s_1. This decision is definitely inconsistent with opponent model O if,

$$\forall u_1 \in s_1 \, \forall u_2 \in s_2 \quad (u_2, u_1) \in O.$$

We can perform this consistency check for every candidate model, for every opponent, for every decision that they made. For each opponent, we eliminate inconsistent models and recompute the generalization of the remaining models to be used in future soft-max^n searches against this opponent.

[5]This is essentially version-space learning [11] where the hypotheses are partial orderings and the training data are decisions the agent has made.

LEMMA 2. *If our opponent is described by some opponent model O_i and it is common knowledge that all players' preferences are consistent with the generalized model $G(O_1, \ldots, O_k)$, then our inference is sound, i.e., we will not eliminate O_i.*

Proof. By Lemma 1 we know that soft-max^n's values for each node in the tree is a superset of all equilibria consistent with the generalized opponent model. Let the opponent's decision for some node in the tree be between set s'_1 and s'_2, where s'_1 was the set that was chosen. Let s_1 and s_2 be the soft-max^n sets for this node. Due to the common knowledge assumption, the soft-max^n sets must be supersets, i.e., $s'_1 \subseteq s_1$ and $s'_2 \subseteq s_2$. For purposes of contradiction, let s_2 strictly dominate s_1 under O_i, thus making the opponent's decision inconsistent with the model. By the definition of strict domination and the superset relation, since s_2 strictly dominates s_1 under O_i, then s'_2 must strictly dominate s_1 and therefore s'_2 must strictly dominate s'_1. Thus all outcomes in s'_2 are preferred to s'_1, and the opponent could not make this decision. $\square$

4.3 Inference Experiments

To show the effectiveness of inference in Spades, we created 3 player types, which we refer to as MT, mOT and ML. As before, the MT player prefers outcomes where more tricks are taken. The mOT player tries to avoid overtricks. The ML player first tries to make their bid. But, among the set of hands where the bid is made or lost, the ML player tries to maximize the number of opponents that miss their bid.

In these games, one player was trying to infer the types of the other players only by observing their decisions. There are 9 ways to combine the 3 opponent types for the two opponents not doing inference, so we played 100 hands for each of these combinations for a total of 900 hands. Over these hands we were able to make 423 inferences about opponent types, although several times we made the same inference about player types multiple times in the same hand.

Because ML can look like both MT and mOT at times, it was the hardest to distinguish. We were only able to rule it out for a player in 51 of the overall hands. We were able to rule out mOT and MT, though, 106 and 103 times respectively. Combined, we successfully made an inference about an opponent in about one in six hands. Correct opponent inferences allow us to use a more specific model in soft-max^n resulting in 4–8% improvements in win percentage based on the results in Tables 2 and 3.3. The average number of hands per game in these results was 15, so this simple notion of consistency is fast enough to allow us to make game-altering inferences about our initially unknown opponents.

4.4 Generalizing Domains

Given that we have based so much of our discussion and experimentation on the game of Spades, it may not be immediately obvious how these results can be generalized. Spades makes a good example because we can easily divide the space of outcomes into two parts – those outcomes in which a player makes their bid, and those outcomes in which they don't. Additionally, it is easy to enumerate all outcomes that can ever occur in a Spades game, and then reason over those outcomes.

But, consider the general case where each player has a utility in the range $\{0...1\}$. In this case there are, in theory, an infinite number of possible outcomes which we would like to reason over. In practice, however, we can reduce the number of possible outcomes using a clustering algorithm, or even more simply, just by dividing the space of possible utilities into discrete divisions. We can then assume that opponents are indifferent between outcomes within each cluster or division.

This is the opposite approach from what is normally taken in two-player zero-sum games. In these games more accurate utilities, often from increased depth of search, have been directly correlated with better play. But, in multi-player games we are seeing the opposite effect. That is, the more accurately we try to model our opponent's utilities, the more likely we are to make a mistake in our modeling. Thus, we want more generic models of our opponent that can tolerate small errors.

Thus, the benefits of soft-max^n may be even more apparent in domains where the number of possible outcomes is larger. Recent work in 4-player chess [9] suggests that max^n does not perform well in this domain because it does not consider minor collaborations which could lead to the max^n player losing the game very quickly. It would be interesting to see if soft-max^n with a generic opponent model could overcome these shortcomings.

5. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed a new algorithm, soft-max^n, for playing multi-player games. Soft-max^n avoids many of the drawbacks of the max^n algorithm, particularly in the face of unknown opponents. Additionally, we have shown how we can use soft-max^n to make inferences about the types of opponents we are playing.

This work forms a very promising foundation for future work in multi-player game-playing. There are still two outstanding issues that must be addressed before soft-max^n can result in human competitive play. The first is to develop less brittle inference. Although we prove the soundness of our technique, human play will not likely correspond to any of our preselected opponent models. In fact, it may not be consistent with *any* opponent model, as it may depend upon external, unobserved circumstances. Hence, our inference could actually eliminate all candidate models in the course of a few hands, leaving no opponent model available for use with soft-max^n. We are currently exploring the replacement of our symbolic reasoning with a more probabilistic approach to mitigate this problem. The second issue is the extension of soft-max^n to imperfect information, which is a common feature of multi-player games. Although Monte-Carlo sampling will allow us to handle the uncertainty during play, the effect of imperfect information on inference still needs to be addressed.

Finally, we are also working to understand the trade-off between the deeper search allowed by max^n pruning algorithms, and the more informed search performed by soft-max^n. It appears that both approaches have their strengths, depending on the situation at hand. Future work is still needed to better understand and classify the relative strengths of each approach.

6. ACKNOWLEDGMENTS

We wish to thank Darse Billings, Martin Müller, Jonathan Schaeffer and Akihiro Kishimoto for their ideas and discussion in relation to this research. This work was partially funded by the Alberta Ingenuity Centre for Machine Learning (AICML) and by Alberta's Informatics Circle of Research Excellence (iCORE).

7. REFERENCES

[1] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M. Bowling, R. Holte, J. Schaeffer, and D. Szafron. Game tree search with adaptation in stochastic imperfect information games. In *Computers and Games*, 2004.

[2] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. In *AAAI-96*, pages 120–125, Aug. 4–8 1996.

[3] H. H. L. M. Donkers, J. W. H. M. Uiterwijk, and H. J. van den Herik. Probabilistic opponent-model search. *Inf. Sci.*, 135(3-4):123–149, 2001.

[4] A. H. M. (Editor), G. Mott-Smith, and P. D. Morehead. *Hoyle's Rules of Games, Third Revised and Updated Edition*. Signet Book, 2001.

[5] M. L. Ginsberg. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 14:303–358, 2001.

[6] H. Iida, J. W. H. M. Uiterwijk, H. J. van den Herik, and I. S. Herschberg. Potential applications of opponent-model search. part 1, the domain of applicability. *ICCA Journal*, 16(4):201–208, 1993.

[7] P. J. Jansen. *Using Knowledge About the Opponent in Game-tree Search*. PhD thesis, Computer Science Department, Carnegie Mellon University, 1992.

[8] R. E. Korf. Generalized game trees. In *IJCAI-89*, pages 328–333, 1989.

[9] U. Lorenz and T. Tscheuschner. Player modeling, search algorithms and strategies in multi player games. In *Advances in Computer Games*, 2005.

[10] C. Luckhardt and K. Irani. An algorithmic solution of N-person games. In *AAAI-86*, volume 1, pages 158–162, 1986.

[11] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[12] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53(2-3):273–290, 1992.

[13] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner. Bayes' bluff: Opponent modelling in poker. In *Proceedings of the Twenty-First Conferece on Uncertainty in Artificial Intelligence*, pages 550–558, 2005.

[14] N. R. Sturtevant. Last-branch and speculative pruning algorithms for max^n. In *IJCAI-03*, pages 669–678, 2003.

[15] N. R. Sturtevant. Leaf-value tables for pruning non-zero sum games. In *IJCAI-05*, pages 317–323, 2005.