Max-Prob: An Unbiased Rational Decision Making Procedure for Multiple-Adversary Environments

Anat Hashavit and **Shaul Markovitch**
Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel
{anatha, shaulm}@cs.technion.ac.il

Abstract

In binary-utility games, an agent can have only two possible utility values for final states, 1 (win) and 0 (lose). An adversarial binary-utility game is one where for each final state there must be at least one winning and one losing agent. We define an unbiased rational agent as one that seeks to maximize its utility value, but is equally likely to choose between states with the same utility value. This induces a probability distribution over the outcomes of the game, from which an agent can infer its probability to win. A single adversary binary game is one where there are only two possible outcomes, so that the winning probabilities remain binary values. In this case, the rational action for an agent is to play minimax. In this work we focus on the more complex, multiple-adversary environment. We propose a new algorithmic framework where agents try to maximize their winning probabilities. We begin by theoretically analyzing why an unbiased rational agent should take our approach in an unbounded environment and not that of the existing *Paranoid* or *MaxN* algorithms. We then expand our framework to a resource-bounded environment, where winning probabilities are estimated, and show empirical results supporting our claims.

1 Introduction

An intelligent agent is an entity that perceives its environment and acts upon it in order to achieve an assigned goal. Which action to take is typically decided by lookahead search in the space of possible world states. The agent uses its perception to determine the current state of the world and evaluates the outcome of alternative action sequences, eventually selecting a sequence it believes will end at a goal state. Adversarial agents operating in the same environment might, however, benefit from preventing the agent from achieving its goal. Therefore, when performing lookahead search, the agent must consider the actions of other agents.

Many algorithms have been developed for various versions of the above problem, mainly for a two-agent adversarial environment, the best known of which is the minimax algorithm [Shannon, 1950], suggested more than half a century ago. Since then, extensive research effort has gone into developing algorithms for decision making in adversarial two-agent environments. But what happens when the number of agents is greater than two?

Ideally, we would expect the known algorithmic solutions for a single-adversary environment to be scalable, and fit the more general domain of a multiple-adversary environment. However, as we will show in the next section, the generalized minimax strategy (or its variation, the *Paranoid* algorithm [Sturtevant and Korf, 2000]), which assumes that all other agents are trying to minimize the agent's utility, is too conservative and might result in a suboptimal decision. While an agent using *Paranoid* assumes any other agent is concerned only with its utility, the *MaxN* algorithm [Luckhart and Irani, 1986] takes the opposite approach and assumes that all other agents are concerned only with their own utility. While this strategy works in general non-cooperative environments, we will show that it also might lead to suboptimal behavior in adversarial setups, where there are dependencies between the agents' utilities.

In this work we define a multiple-adversary environment where the utility of a final state for an agent can be either 1(win) or 0(lose). We then define an agent that seeks to maximize its utility value, but is equally likely to choose between states with the same utility value. We call such an agent an unbiased rational agent. We show that a multiple-adversary environment with unbiased rational agents induces a probability distribution over the possible outcomes of the game. We claim that an unbiased rational agent acting in this environment should strive to maximize its winning probability. We then present a new algorithm, Max-Prob, for unbiased rational lookahead-based decision making in a multiple-adversary environment and show that existing algorithms may act suboptimally in such a setup. In addition, we present a variation of the algorithm for a resource-bounded setup and empirically show its advantage over alternative approaches.

2 Problem Analysis: Unbounded Unbiased Rational Agents

The process where multiple agents commit actions in the environment in order to achieve their assigned goal is often referred to as a game. We define a *binary-utility game* as one consisting of the following elements:

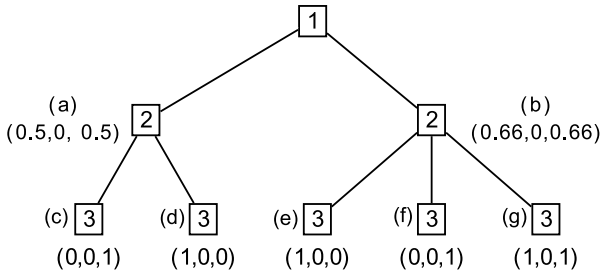

Figure 1: *Max-Prob*. Example of a complete tree.

1. A set of agents $A = \{a_1, ..., a_n\}$.
2. A set of game states, S.
3. An initial state $s_i \in S$.
4. A set $F \subseteq S$ of final states.
5. A function, Turn : $S \rightarrow A$, which designates the acting agent in the given state. We assume that only one agent acts at each state.
6. A function, Succ : $S \rightarrow 2^S$, which returns all the states directly succeeding Turn(s), under the constraints of the game, s.t $\forall n,\ s \notin Succ(s)^n$, where $Succ(s)^n$ denotes the set of states reachable from s after a sequence of n turns.
7. A binary utility function, $U : F \times A \rightarrow \{0, 1\}$, which returns, for an agent A_i and a final state $s_f \in F$, a binary score, denoting whether the agent has won.

We further define an *adversarial binary-utility* game as one where for each $f \in F$:

$$|A| > |\{a \in A | U(f, a) = 0\}| > 0.$$

This condition, that at each state there is at least one winner and one loser, is a minimal condition for conflict between agents.

We call a state $s \in S$ that satisfies Succ$(s) \subseteq F$ a *pre-final* state. We further define for such a state the set of winning states as $W(s, a) = \{s' \in \text{Succ}(s) | U(s', a) = 1\}$, and similarly the set of losing states $L(s, a)$. We define an *unbiased rational* agent as one that seeks to maximize its utility value but is equally likely to choose between states in which it has the same utility value. In a *pre-final* state, such an agent selects a state from $W(s, a)$ with equal probability if $W(s, a)$ is not empty, and selects one from $L(s, a)$ with equal probability otherwise.

In this work we focus on *multiple-adversary* games where $n > 2$. In such an environment, unbiased rational agents induce a probability distribution over the possible outcomes of the game. This is shown in figure 1. At node a for example, agent 2 will choose between nodes c or d with equal probability, causing the game to have two possible outcomes, each with a probability of 0.5.

Given the above definitions, in an environment consisting of unbiased rational agents with unbounded resources, each agent should pick a move that maximizes its probability to win and, if several such moves exist, to choose one randomly with uniform distribution.

The probability of an agent a to win at a state s, $P_w(s, a)$, is now recursively defined, under the assumption that all agents are unbiased rational agents:

$$P_w(s,a) = \begin{cases} U(s,a) & s \in F \\ & (a = Turn(s) \\ {}^1Max\{P_w(s',a) | s' \in Succ(s)\} & \text{and } s \notin F) \\ \sum_{s' \in C_{max}(s)} P_w(s',a) * \frac{1}{|C_{max}(s)|} & s \notin F \end{cases}$$

where

$$C_{max}(s) = \arg \max_{s' \in succ(s)} P_w(s', Turn(s)).$$

The *Max-Prob* algorithm for an unbounded unbiased rational agent, $a \in A$, computes $P_w(s', a)$ for all successors of the current state and selects the move leading a to a state with maximal P_w. If several such moves exist, it selects one randomly.

A running example of the algorithm on a complete tree can be seen in Figure 1. There are three agents playing, with agent 1 invoking the *Max-Prob* procedure. The vectors at the nodes represent the P_w values of all three agents. At node a, agent 2 is the one to move. Since it cannot win at this state and is indifferent as to which of the other two agents will, it randomly selects either node c or d. Therefore, the winning probability of agents 1 and 3 is 0.5. Similarly, the winning probabilities for node b are 0.66, 0 and 0.66 for agents 1, 2, 3 respectively. Therefore, according to *Max-Prob*, agent 1 will select the move leading it to node b, where its probability of winning is higher.

Note that, unlike other algorithms such as *MaxN*, where every value in a mid-game tree node coincides with a value of a leaf node, our algorithm can have mid-game tree-node values that do not coincide with any of the leaf nodes, for example, nodes a and b.

The expected utility value of agent 1 is equal to its probability to win the game. Agent 2 is an unbiased rational agent. It will choose node c with a probability of 0.5, and either node e or node g with a probability of $\frac{1}{3}$. The expected utility value of agent 1 will therefore depend on the probability of it choosing between nodes a and b.

The generalized minimax algorithm (and *Paranoid*) will propagate a value of 0 to both nodes a and b, and will then select one randomly. This means that any tie-breaking rule that will not strictly choose node b will cause sub-optimal play in an environment of unbiased rational agents. For the case of a uniformly-distributed tie-breaker, the agent will sub-optimally choose node a, with probability 0.5. The expected utility value of *Paranoid* in this case will be $\frac{7}{12}$.

MaxN will randomly select the values of one of the two nodes, either c or d, to propagate to node a and similarly one of e, f, or g to propagate to node b. This means that there are six possible propagation combinations. In two of them the agent will select b, in one it will select a, and in the other three it will select randomly. The algorithm's performance will again depend on the tie-breaker. But even if the agent will use a uniformly-distributed tie-breaker like the one that

[1]This value can also be computed by the calculation presented for the general case.

unbiased rational agents use, it will still sub-optimally choose a with a probability of $\frac{5}{12}$. The expected utility value of *MaxN* in this case will be approximately 0.6.

Max-Prob will always select node b. Its expected utility will therefore be 0.66, higher than that of both *MaxN* and *Paranoid*.

3 The *Max-Prob* Algorithm: Bounded Agents

It is well known that expanding the complete game tree is computationally impossible for most if not all games. Thus, a partial game tree is usually explored. In this section we will show a version of the *Max-Prob* algorithm that is suitable for bounded unbiased rational agents. As a bounded unbiased rational agent usually cannot expand the search tree down to the level of the final states (the leaves of the game tree), our challenge is to find a way to estimate the P_w values of internal game-tree nodes that are leaves of the search tree.

Our solution is based on traditional heuristic evaluation functions, but with an additional requirement. We define a trait of heuristic functions for games, called *win-distinguishing*, and restrict our algorithm to use only such heuristic functions. A *win-distinguishing* heuristic function is one that at the end of the game assigns the winners of the game, and only them, an equal value that is higher than that of all other agents.

Definition 1 (win-distinguishing) *Let $a_i \in A$ be an agent. A heuristic function $h^{a_i} : S \longrightarrow \mathbb{R}$, is* win-distinguishing *iff* $\forall s \in F,\ U(s, a_i) = 1 \Leftrightarrow h^{a_i}(s) \geq h^{a_j}(s)\ \forall i \neq j$.

Our goal is to use heuristic estimation to estimate P_w, the probability of winning the game. If we use a *win-distinguishing* heuristic function, then the winning probability of an agent a at a state s is the probability of this agent to have the highest heuristic value at the end of the game.

Let $s \in S$ be a game state. We denote the set of all final states in F that are reachable from s as F_s. Under the assumption of unbiased rational play defined in the previous section, there is a probability distribution P_{F_s} over F_s that determines for each final state $f \in F_s$ the probability of reaching it under unbiased rational play from state s. Given a heuristic function h^a, we denote by $f^s_{h^a}$ the probability density function describing the distribution of h^a's values over the set of all final states $F'_s = \{f \in F_s | P_{F_s}(f) > 0\}$.

For practical reasons, we assume for now that $f^s_{h^a}$ is uniform over an interval $[l_{h^a}(s), u_{h^a}(s)]$. In that case the winning probability of each agent is estimated by the accumulation of winning probabilities for each possible value in the range $[l_{h^a}(s), u_{h^a}(s)]$, as described in the following equation [2]

$$\hat{P_w}(s,a) = \int_{x=l_{h^a}(s)}^{u_{h^a}(s)} f^s_{h^a}(x) \cdot \prod_{a' \in A \setminus \{a\}} \int_{y=-\infty}^{x} f^s_{h^{a'}}(y) dy\, dx. \quad (1)$$

[2] Ideally, we would estimate the probability that h_a is maximal for an arbitrary leaf $f \in F_s$. Since this probability is too difficult to estimate, we instead estimate the win probability by comparing the expected h values (or their bounds) for the agents.

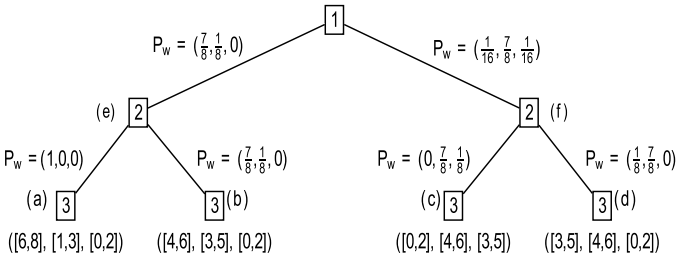

Figure 2: *Max-Prob*. Partial tree evaluation.

The internal integral evaluates $P(y \leq x)$, where y is the value associated with agent a' and x is the value associated with agent a. For a uniform distribution this value is 1 when $u_{h^{a'}}(s) < l_{h^a}(s)$ and 0 when $u_{h^a}(s) < l_{h^{a'}}(s)$ (thus zeroing the whole expression); otherwise it is $\frac{x - l_{h^{a'}(s)}}{u_{h^{a'}}(s) - l_{h^{a'}}(s)}$.

Considering the above observations, we suggest an adaption of the algorithm presented in the previous section. The algorithm will remain the same except for the leaf evaluation method, which will first produce a vector of intervals bounding the heuristic value for each agent, and then infer from these intervals the probability of each agent to win according to Equation 1. The values propagated up the tree will be the estimated winning probabilities. The algorithm is listed in Figure 3.

An example of how *Max-Prob* assigns values in a partial game tree is presented in figure 2. At node (a) the value of agent 1 will always be higher than that of all the other agents. Therefore, it will have a winning probability of 1 while all the other agents will have a winning probability of 0. Each of the other leaf nodes contain intervals that intersect with each other. We will need to compute the winning probabilities, according to the rules of uniform distribution and Bayes' theorem. At node b, for example, agent number 3 has no chance of being the winner and is quickly taken out of the equation. For the other two agents we will need to examine the possible heuristic values of the final states in F'_s. For a state f where $h^{a_2}(f) \in [3,4]$, a_1 has probability of 1 to win, and again for the case where $h^{a_2}(f) \in [4,5]$, and $h^{a_1}(f) \in [5,6]$. If $h^{a_2}(f) \in [4,5]$ and $h^{a_1}(f) \in [4,5]$, a_1 has probability of 0.5 to win. The probabilities of each of these scenarios to occur are 0.5, 0.25, 0.25 respectively, and so

$$\hat{P_w}(b, a_1) = 0.5 * 1 + 0.25 * 1 + 0.25 * 0.5 = 0.875.$$

The value for a_2 can be similarly computed, and so can the values at nodes c and d. The winning probabilities are then propagated up the tree according to the *Max-Prob* algorithm. At node e agent 2 has only one maximal child, but for node f both children maximize the agent's winning probability and so the expected probability values propagated for agents 1 and 3 are not identical to either of their values at the leaf nodes.

4 Empirical Evaluation

4.1 Experimental Methodology

We compared the *Max-Prob* algorithm with *MaxN* and *Paranoid*. In addition, we compared it to the recently developed

Procedure MAX-PROB(s, a)
$C \leftarrow Succ(s)$
$P_w \leftarrow Max\{\text{WinProb}(c)[a] \mid c \in C\}$
$C_{max} \leftarrow \{c \in C | \text{WinProb}(c)[a] = P_w(s)\}$
Return $rand(C_{max})$

Procedure WINPROB(s)
If $s \in F$
Foreach $a \in A$
$P_w[a] = \hat{P}_w(s, a)$
Return P_w
$C \leftarrow Succ(s)$
$curr = Turn(s)$
$P_w[curr] \leftarrow Max\{\text{WinProb}(c)[curr] | c \in C\}$
$C_{max} \leftarrow \{c | c \in C, \text{WinProb}(c)[curr] = P_w[curr]\}$
Foreach $a \in A \backslash curr$
$P_w[a] \leftarrow \sum_{c \in C_{max}} \frac{1}{|C_{max}|} P_w(c)[a]$
Return P_w

Figure 3: *Max-Prob*. Algorithm description.

MP-Mix algorithm [Zuckerman *et al.*, 2009], which alternates between playing *MaxN*, *Paranoid*, and an offensive strategy that attacks the leading agent. It does so by examining the difference between the heuristic value of the leading agent and that of the runner up. If the agent is the leading agent and the said difference is higher than a predefined threshold, T_d, a paranoid strategy is taken. Otherwise, if the leading agent is another agent and this difference is higher than a second threshold, T_o, an offensive strategy is taken. In all other cases, the *MaxN* strategy is played.

The experiments were conducted on four different games, two board games and two card games. For the card games setup, we used the perfect information version of the games. The depth of the search tree was limited to 8 plies, and we did not allow the search to end in the middle of a trick. Algorithm performance was compared on 100 different hands that were played for each of the 24 possible player orderings using two different setups. In the first setup, we ran 100 single hand games for each player ordering, and in the second a single 100 hands game for each player ordering. An agent was rewarded with one victory point for each game where it outperformed its opponents. Since these are not single winner games, several agents could gain a victory point in a single game.

The board games have a much higher branching factor than the card games and so their setup was slightly different. Each game tested three of the four competing algorithms, and the depth of the tree was limited to 4.

The games we used are:

1. *Perfect-information Simplified Spades* Spades is a trick taking card game for which the spades suite is always a trump. The game has both a partnership and a solo version; we will focus on the solo version. The goal of the game is to have the highest score among all players. In the original version of the game, the scoring is affected by the number of tricks a player took and the bid she placed at a preliminary phase. Our simplified version omits the bidding phase and awards 10 points for every trick taken. The heuristic function we used was constructed from the number of tricks taken and the composition of the player's hand.
2. *Perfect-information Hearts* Hearts is another hidden information trick taking card game. Each heart card taken is worth 1 penalty point and the queen of spades is worth 13. The goal of the game is to have the lowest number of penalty points at the end of the game. Our heuristic function consisted of the number of penalty points taken by the players and the composition of the players' hand.
3. *ChineseCheckers* Chinese Checkers is an abstract strategy game that can be played with up to six players. The goal of the game is to be the first player who moves all ten of her pawns from her home camp, located at one pit of a hexagram, to her destination camp, located at the opposite pit. Our heuristic was simple and consisted of the accumulated distance of the pawns from their destination camp.
4. *Abalone* Abalone is another abstract strategy game. Its board is an hexagon containing 61 slots. In the three player version which we play, each player starts the game with eleven marbles arranged in two adjunct rows, one containing 5 marbles and another containing 6. The goal of the game is to be the first player who manages to push six of her opponents' marbles out of the game board. The heuristic function we used contained the number of opponents' pawns pushed off the board and the pawns' arrangement on the board.

Game	depth	*Max-Prob*	*MaxN*	*Paranoid*	*MP-Mix*
Chinese Checkers	4	52	21.5	24.5	2
Abalone	4	45.7083	5	44.0417	5.25
Spades	8	26.39	25.93	25.63	22.05
Hearts	8	31	26	18	25
Tournament Spades	8	50	16.7	33.3	0
Tournament Hearts	8	75	25	0	0
Average win percentage		**46.68305**	20.02166667	24.24528333	9.05

Table 1: Winning percentage for each game

Each of our heuristic functions can be bounded by a constant number. We used these numbers as the upper bounds for the *Max-Prob* intervals and the current heuristic value of a state as the lower bound. Tie-breaking was done only at the root level, where we preferred states with a higher heuristic value. For the *MP-Mix* algorithm we set the thresholds to values that indicate a notable improvement in an agent's status: a trick taken for Spades, a penalty point for Hearts, pushing another agent's pawn off the board for Abalone, and a three slots distance gap for Chinese Checkers.

4.2 Results

Table 1 presents the percentage of games won by each player. It can be seen that the *Max-Prob* algorithm outperforms its opponents with an average win percentage that is more than 20% higher than that of the runner up *Paranoid*, which had

better results than both *MaxN* and *Mp-Mix*. It would appear that, given no other options, playing according to a paranoid assumption will bring better results than playing according to the indifference assumption of *MaxN*.

Although T_d and T_o were set to values which reflected a genuine advantage in the games used for testing, *Mp-Mix* ranked last in most of the domains tested. These results are inconsistent with the results presented in the original paper. Therefore, except for the need to perform a parameter tuning process before competing *Mp-Mix* with other algorithms, we cannot draw any conclusions regarding its performance in relation to *Max-Prob*. We do believe that with additional parameter tuning, for each of the domains tested, *Mp-Mix* can achieve much better results.

For the board games, we did a lot better in Chinese Checkers than in Abalone. In the card games, *Max-Prob* achieved the best results for tournaments, even though scoring of previous games was taken into account in the heuristic function, which was equal for all agents.

The goal of each of the games we tested was to minimize or maximize some metrics in respect to the other agents. The heuristic functions were constructed from two components, a direct one, which contained the current value of the metrics, and an indirect one, which attempted to estimate any future change in metric values. We can see that *Max-Prob* performs better in setups where the indirect part has less weight.

In Chinese Checkers, where *Max-Prob* wins more than half the games, we minimize the accumulated distance of the pawns from their home camp, and this metric is the only component of the heuristic function. In the card games we used heuristic functions that contained both a direct and an indirect component. The direct component counted the number of tricks or penalty points of a player and the indirect component used the composition of the player's hand in order to estimate future changes to these metrics. Max-Prob had less of an advantage in the single-hand card games than in Chinese Checkers. In the tournament setups, however, the performance improved notably. This is due to the diminishing effect of the indirect parts of the heuristics as more games are played. In Abalone, an opponent's pawn is pushed off the board in a small percentage of the moves, and so the indirect component of the heuristic was the one affecting the win probability calculation for most of the game. This made it difficult for *Max-Prob* to accurately estimate the winning probability, as reflected by the results.

5 Related Work

Luckhart and Irani [1986] presented the *MaxN* algorithm. In this algorithm, evaluation of a leaf in the game tree produces a vector of N values. Each component in the vector is the utility value of the corresponding player for that state. The algorithm also defines a propagation rule which states that each agent in its turn selects a move maximizing it individual utility value, without considering the utility values of the other players. This means that there is an unstated assumption that the agents are indifferent to each other's performance, an assumption that does not hold for adversarial games.

Sturtevant [2000] proposed the *Paranoid* algorithm which generalizes the classical minimax algorithm from a two-player game into an N-players game by assuming that a coalition of N-1 players has been formed to play against the remaining player. It is "paranoid" in the sense that it assumes collaboration against a single player. Unlike algorithms that preserve the N player structure of the game, *Paranoid* can use deep pruning, as shown by Korf [1991]. This is a distinct advantage of the algorithm. However, the paranoid assumption can lead to suboptimal play [Sturtevant and Korf, 2000]. Furthermore, using *Paranoid* renders all individual opponent modelings unusable, even if these are available.

Zuckerman et al. [2009] presented the *MP-Mix* algorithm, which implements a mixed strategy approach. When it is an agent's turn to act, it examines the difference between the leading agent's heuristic value and that of the runner up. It then decides accordingly whether to use an offensive strategy, a paranoid strategy, or the default *MaxN* strategy. This algorithm overcomes *MaxN*'s inherent problem of the agents' indifference to each other by explicitly examining the relation between the leading agent and the runner up. However, this solves the indifference problem only in part since it does not take into account the status of the other agents, whereas *Max-Prob* does so implicitly in the calculation of the winning probabilities.

Probabilistic methods, and methods which use ranges of values, have been used mainly as a tool for better incorporating opponent models in the search algorithm, or for the purpose of selective deepening. Most such methods were implemented only in single-adversary environments. The B^* algorithm proposed by Berliner [1979] defines for each node an interval, bounded by an optimistic and a pessimistic value. These intervals are then used to selectively expand nodes. The interval values of a node are derived from its children's values and are backed up the tree, in an attempt to narrow the range of the root's interval until it is reduced to a single value.

Palay [1985] extended the B^* algorithm to a version where the evaluation of a leaf produces probability distributions, which are then propagated up the tree using the product propagation rule. Baum and Smith [1997] proposed a Bayesian framework to selectively grow the game tree and evaluate tree nodes. Their algorithm, called BP, describes how to evaluate the uncertainty regarding the evaluation function. Instead of assigning single values to nodes, discrete distributions on the possible values of the leaves are assigned and propagated up the tree, again using the product propagation rule. These algorithms are similar to the *Max-Prob* algorithm in that the leaf evaluation procedure does not produce a single value. But they continue to propagate nondiscrete values, unlike *Max-Prob*, which propagates vectors of discrete winning probabilities deduced from the evaluation of the leaves.

The M_ϵ^* algorithm [Carmel and Markovitch, 1996] deals with an agent that is uncertain as to the evaluation function of its opponents and thus tries to contain it within an error bound, similar to the *Max-Prob* algorithm. Unlike the *Max-Prob* algorithm, however, M_ϵ^* does not convert these intervals to single values but rather propagates intervals of values. Donkers [2001] proposes *PrOM*, an extension to opponent modeling search that takes into account uncertainties in opponent models and allows several models to be considered,

with different probabilities assigned to each. Sturtevant and Bowling [2006] present the *Soft-maxn* algorithm, an extension of *MaxN*, that propagates sets of *MaxN* vector values instead of choosing just one, in case of a tie or several possible opponent models. The *Soft-MaxN* concept was later generalized for the case where a probability distribution is known for the candidate opponent models in the *Prob-MaxN* algorithm [Sturtevant *et al.*, 2006]. This method combines the idea of *Soft-MaxN* to propagate sets of vectors and the probabilistic opponent model search for two players proposed by Donkers.

Methods which propagate probabilities not for the sake of selective deepening or as part of an opponent modeling algorithm were discussed only in the domain of two player games. Pearl [1984] suggested assigning winning probabilities to the leaves and propagating them using the product rule. He also suggested a way to compute these probabilities using statistical records for chess, later investigated by Nau [1983] for two-player games.

6 Conclusions

We presented a new framework for lookahead-based decision making in a multiple-adversary environment. Our framework defined unbiased rational agents and the way such agents should act for both bounded and unbounded resource environments. We validated our framework by theoretical analysis and empirical evaluations.

References

[Baum and Smith, 1997] Eric B. Baum and Warren D. Smith. A Bayesian approach to relevance in game playing. *Artif. Intell.*, 97, 1997.

[Berliner, 1979] Hans Berliner. The b* tree search algorithm: A best-first proof procedure. *Artif. Intell.*, 12(1):23 – 40, 1979.

[Carmel and Markovitch, 1996] David Carmel and Shaul Markovitch. Learning and using opponent models in adversary search. Tech. Report CIS9609, Technion, 1996.

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

[Korf, 1991] Richard E. Korf. Multi-player alpha-beta pruning. *Artif. Intell.*, 48(1):99 – 111, 1991.

[Luckhart and Irani, 1986] Carol Luckhart and Keki B. Irani. An algorithmic solution of n-person games. In *AAAI*, pages 158–162, 1986.

[Nau, 1983] Dana S. Nau. Pathology on game trees revisited, and an alternative to minimaxing. *Artif. Intell.*, 21(1-2):221–244, 1983.

[Palay, 1985] Andrew J. Palay. *Searching with probabilities*. Pitman Publishing, Inc., Marshfield, MA, 1985.

[Pearl, 1984] Judea Pearl. *Heuristics: Intelligent Search Strategies For Computer Problem Solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[Shannon, 1950] C.E. Shannon. Programming a computer for playing chess. *Phil. Mag.*, 41:256–275, 1950.

[Sturtevant and Bowling, 2006] Nathan Sturtevant and Michael Bowling. Robust game play against unknown opponents. In *Proc. of AAMAS '06*, pages 713–719, 2006.

[Sturtevant and Korf, 2000] Nathan R. Sturtevant and Richard E. Korf. On pruning techniques for multi-player games. In *AAAI/IAAI*, pages 201–207, 2000.

[Sturtevant *et al.*, 2006] Nathan R. Sturtevant, Martin Zinkevich, and Michael H. Bowling. Prob-maxn: Playing n-player games with opponent models. In *AAAI*, 2006.

[Zuckerman *et al.*, 2009] Inon Zuckerman, Ariel Felner, and Sarit Kraus. Mixing search strategies for multi-player games. In *Proc. of IJCAI'09*, pages 646–651, 2009.