The MP-MIX Algorithm: Dynamic Search Strategy Selection in Multiplayer Adversarial Search

Inon Zuckerman and Ariel Felner

Abstract—When constructing a search tree for multiplayer games, there are two basic approaches to propagating the opponents’ moves. The first approach, which stems from the MaxN algorithm, assumes each opponent will follow his highest valued heuristic move. In the second approach, the paranoid algorithm, the player prepares for the worst case by assuming the opponents will select the worst move with respect to him. There is no definite answer as to which approach is better, and their main shortcoming is that their strategy is fixed. We therefore suggest the MaxN-paranoid mixture (MP-Mix) algorithm: a multiplayer adversarial search that switches search strategies according to the game situation. The MP-mix algorithm examines the current situation and decides whether the root player should follow the MaxN principle, the paranoid principle, or the newly presented directed offensive principle. To evaluate our new algorithm, we performed extensive experimental evaluation on three multiplayer domains: Hearts, Risk, and Quoridor. In addition, we also introduce the opponent impact (OI) measure, which measures the players’ ability to impede their opponents’ efforts, and show its relation to the relative performance of the MP-mix strategy. The results show that our MP-mix strategy significantly outperforms MaxN and paranoid in various settings in all three games.

Index Terms—Artificial intelligence (AI), decision trees, game-tree search, multiplayer games.

I. INTRODUCTION

From the early days of artificial intelligence (AI) research, game playing has been one of the prominent research directions, since outplaying a human player has been viewed as a prime example of an intelligent behavior which surpasses human intelligence.

The main building block of game playing engines is the adversarial search algorithm, which defines a search strategy for the action selection on the possible actions a player can take. In general, two-player adversarial search algorithms have been an important building block in the construction of strong players, sometimes optimal or world champions [4], [13]. Classical two-player adversarial search algorithms include the minimax search algorithm coupled with the alpha-beta pruning technique [6] which is still the basic building block for many successful computer player implementations. In addition, over the years many other variations of the original algorithm have been suggested [12].

When constructing a search tree for multiplayer games, there are two basic approaches one can take when expanding the opponents’ moves. The first approach, presented in the MaxN algorithm [9], assumes that each opponent will follow his highest valued move. In the second approach, presented in the paranoid algorithm [16], the player prepares for the worst case by assuming the opponents will work as a coalition and will select the worst move with respect to him.

A comprehensive comparison between the two algorithms was performed by Sturtevant [14]. This could not conclude a definite answer as to which approach is better, and further claims that the answer strongly depends on the properties of the game and on the evaluation function. The main weakness of these algorithms is that their underlying assumptions on opponents’ behavior is fixed throughout the game. However, when examining the course of many games one can realize that neither of their underlying assumptions are reasonable for the entire duration of the game. There are situations where it is more appropriate to follow the MaxN assumption, while in other situation the paranoid assumption seems to be the appropriate approach.

Our focus in this work is on multiplayer games with a single winner and no reward is given to the losers, i.e., they are all equal losers regardless of their losing position. We call these games single-winner games. In such multiplayer games, there naturally exist other possible approaches to propagate heuristic values, that is besides MaxN and paranoid. In this paper, we introduce one such new approach, denoted the offensive strategy. In single-winner, multiplayer games, there are situations where one player becomes stronger than the others and advances towards a winning state. Such situations, together with an understanding that there is no difference whether a loser finishes second or last (as only the winner gets rewarded), should trigger the losing players to take explicit actions in order to prevent the leader from winning, even if the actions temporarily worsen their own situation. Moreover, in some situations, the only way for individual players to prevent the leader from winning is by forming a coalition of players. This form of reasoning should lead to a dynamic change in the search strategy to an offensive strategy, in which the player selects the actions that worsen the situation of the leading player. At the same time, the leading player can also understand the situation and switch to a more defensive strategy and use the paranoid approach, as its underlying assumption does reflect the real game situation.
All these approaches (MaxN, paranoid, and offensive) are fixed. We introduce the \textit{MaxN-paranoid mixture} (MP-mix) algorithm, a multiplayer adversarial search algorithm which switches search strategies according to the game situation. MP-mix is a meta-decision algorithm that outputs, according to the players’ relative strengths, whether the player should conduct a game-tree search according to the MaxN principle, the paranoid principle, or the newly presented directed offensive principle. Thus, a player using the MP-mix algorithm will be able to change his search strategy dynamically as the game develops. To evaluate the algorithm we implemented the MP-mix algorithm on three single-winner and multiplayer domains:

1. \textit{Hearts}—an imperfect-information, deterministic game;
2. \textit{Risk}—a perfect-information, nondeterministic game;
3. \textit{Quoridor}—a perfect-information, deterministic game.

Our experimental results show that in all domains, the MP-mix approach significantly outperforms the other approaches in various settings, and its winning rate is higher. However, while the performance of MP-mix was significantly better in \textit{Risk} and \textit{Quoridor}, the results for \textit{Hearts} were less impressive.

In order to explain the different behavior of MP-mix we introduce the opponent impact (OI). The OI is a game specific property that describes the impact of move decisions by a single player on the performance of other players. In some games, the possibilities to impede the opponents are limited. Extreme examples are the multiplayer games of \textit{Bingo} and \textit{Yahtzee}. In other games, such as \textit{Go Fish}, the possibilities to impede the opponent almost always exist. We show how OI can be used to predict whether dynamically switching the search strategies and using the MP-mix algorithm are beneficial. Our results suggest a positive correlation between the improvement of the MP-mix approach over previous approaches and games with high OI.

The structure of the paper is as follows. Section II provides the required background on the relevant search techniques. In Section III, we present the newly suggested directed offensive search strategy and the MP-mix algorithm. Section IV presents our experimental results in three domains. The OI is introduced and discussed in Section V. We conclude in Section VI and present some ideas for future research in Section VII. This paper extends a preliminary version that appeared in [19].

II. BACKGROUND

When a player needs to select an action, he spans a search tree where nodes correspond to states of the game, edges correspond to moves, and the root of the tree corresponds to the current location. We refer to this player as the root player. The leaves of the tree are evaluated according to a heuristic static evaluation function (will be shortened to evaluation function from now on) and the values are propagated up to the root. Each level of the tree corresponds to a different player and each move corresponds to the player associated with the outgoing level. Usually, given $n$ players the evaluation function gives $n$ values, each of them measures the merit of one of the $n$ players. The root player chooses a move towards the leaf whose value was propagated all the way up to the root (usually denoted the principal leaf). When propagating values, the common assumption is that the opponents will use the same evaluation function as the root player (unless using some form of specific opponent modeling-based algorithm such as the ones found in [1] and [17]).

In sequential two-player zero-sum games (where players alternate turns), one evaluation value is enough, assuming one player aims to maximize this value while the other player aims to minimize it. The evaluation value is usually the difference between the merits of the max player and the merit of the min player. Values from the leaves are propagated according to the well-known minimax principle [18]. That is, assuming the root player is a maximizer, in even (odd) levels, the maximum (minimum) evaluation value among the children is propagated.

Sequential, turn-based, multiplayer games with \( n > 2 \) players are more complicated. The assumption is that for each node the evaluation function returns a vector $H$ of $n$ evaluation values where $h_i$ estimates the merit of player $i$. Two basic approaches were suggested to generalize the minimax principle to this case: MaxN [9] and paranoid [16].

A. MaxN

The straightforward and classic generalization of the two-player minimax algorithm to the multiplayer case is the MaxN algorithm [9]. It assumes that each player will try to maximize his own evaluation value (in the evaluation vector), while disregarding the values of other players. Minimax can be seen as a special case of MaxN, for \( n = 2 \). Fig. 1 (taken from [14]) presents an example of a three-player search tree, alongside the evaluation vector at each level while activating the MaxN algorithm. The numbers inside the nodes correspond to the player of that level. The evaluation vector is presented below each node. Observe that the evaluation vectors in the second level were chosen by taking the maximum of the second component while the root player chooses the vector which maximizes his own evaluation value (the first component). In this example, the root player will eventually select the rightmost move that will resolve in node $c$ as he has the highest evaluation value ($\{=6\}$) for the root player.
III. COMBINING SEARCH APPROACHES

Given the MaxN and the paranoid multiplayer adversarial search algorithms, which one should a player use? As there is no theoretical or experimental conclusive evidence revealing which approach is better, our intuitive underlying hypothesis (inspired from observing human players) is that the question of which search algorithm to use is strongly related both to the static properties of the games that are derived from their rules, and to dynamic properties that develop as the game progresses. It might be that in the same game, in some situations, it would be worthwhile using the MaxN algorithm while in other cases the paranoid would be the best approach. For that, we suggest the MP-mix decision algorithm that dynamically chooses which approach to use based on these attributes.

Before continuing with the technical details we would like to illustrate the intuition behind MP-mix in the strategic board game Risk (we will provide a detailed game description in Section IV-B). In early stages of the game, players tend to expand their borders locally, usually trying to capture a continent and increase the bonus troops they receive each round. In advance stages, one player might become considerably stronger than the rest of the players (e.g., he might control three continents which will give him a large bonus every round). The other players, having the knowledge that there is only a single winner, might understand that regardless of their own individual situation, unless they put some effort into attacking the leader, it will soon be impossible for them to prevent the leading player from winning. Moreover, if the game rules permit, the weak players might reach an agreement to form a temporary coalition against the leader. In such situations, the strongest player might understand that it is reasonable to assume that everybody is against him, and switch to a paranoid play (which might yield defensive moves to guard its borders). In case the situation changes again and this player is no longer a threat, it should switch its strategy again to its regular self-maximization strategy, namely MaxN.
A. The Directed Offensive Search Strategy

Before discussing the MP-mix algorithm we first introduce a new propagation strategy called the directed offensive strategy (denoted offensive) which complements the paranoid strategy in an offensive manner. In this new strategy, the root player first chooses a target opponent he wishes to attack. He then explicitly selects the path which results in the lowest evaluation score for the target opponent. Therefore, while traversing the search tree, the root player assumes that the opponents are trying to maximize their own utility (just as they do in the MaxN algorithm), but in his own tree levels he selects the lowest value for the target opponent. This will prepare the root player for the worst case where the opponents are not yet involved in stopping the target player themselves.

Fig. 3 shows an example of a three-player game tree, in which the root player runs a directed offensive strategy targeted at player 3 (labeled $3^t$). In this case, player 2 will select the best nodes with respect to his own evaluation (ties are broken to the left node), and the root player will select to move to node $c$ as it contains the lowest value for player $3^t$ (as $0 < 2$).

As stated above, if coalitions between players can be formed (either explicitly via communication or implicitly by mutual understanding of the situation), perhaps several of the opponents will decide to join forces in order to “attack” and counter the leading player, as they realize that it will give them a future opportunity to win. When this happens, the root player can run the same offensive algorithm against the leader but under the assumption that there exists a coalition against the leader which will select the worst option for the leader and not the best for himself.

B. Pruning Techniques

A number of pruning techniques that generalize alpha–beta for two player games are applicable in multiagent games. In order to achieve some sort of pruning in multiplayer games we need the following conditions to hold [8]:

1) the evaluation function must have an upper bound on the sum of all the components of the evaluation vector;
2) a lower bound on the value of each component exists.

These requirements are not very limited as most practical heuristic functions satisfy these conditions. For example, a fair evaluation function for multiplayer Othello (the formal four-player version is called Rolit) will count the number of pawns the player currently has on the board. This number will have a lower bound of 0 and an upper bound of the number of board squares, namely 64. Thus, both requirements hold.

We now present the three types of pruning procedures that are part of the alpha–beta pruning for two-player games and discuss which pruning is applicable for the offensive search strategy.1

1) Immediate Pruning: This is the simplest and the most intuitive type of pruning. Assume that it is the root player’s turn to move, that $i$ is the target player, and that the $i$th component of one of the root player’s children equals the minimal possible evaluation value. In this case, he can prune the rest of the children as he cannot get a value which will be worse for player $i$. When we simulate action selection in opponent levels (i.e., all levels excluding the root player’s level), immediate pruning can prune all children when the player has the maximal possible value for his component in the tuple. For example, in the tree presented in Fig. 4, with heuristic function values in the $[0, 10]$ range, the right node was pruned by the root player since the middle node already presented the minimal value for the target player.

2) Failure of Shallow Pruning in the Offensive Strategy: As stated above, Korf showed that only limited shallow pruning is applicable in MaxN [16]. We now show that shallow pruning is not applicable in the tree level following the offensive search player. Even though we can restrict the upper bound on the target player’s score, since we are interested in minimizing its value we cannot conclude whether the real value is above or below the current value. Thus, the bound is useless. Let us illustrate the matter with the following example (Fig. 5), where player 3 is the target player. The left branch returned a value of 5 from node (a), thus, at the root we can mark 5 as a new upper bound for the target’s score and, as the functions sum to 10 we can conclude $10 - 5 = 5$ as upper bounds for player 1 and player 2. Moving to node (b), we attain 2 as the value for player 2, and we can conclude that players 1 and 3 have at most a score value of $10 - 2 = 8$. Now, player 1 cannot prune the rest of (b)’s children as he does not know if the actual value is lower or higher than the current bound 5. It is possible to prune only if we know the actual value of each position in the tuple. It

1We adapt the same terminology for naming the different pruning procedures as found in [8].
is important to add that shallow pruning might be applicable in the levels of the maximizing players, that is, between players 2 and 3 and players 3 and 1.

3) Deep Pruning: The third and most important type of pruning is deep pruning where we prune a node based on the value we receive from its great-grandparent or any other more distant ancestor. It has already been shown that deep pruning is not possible in MaxN [16] and for the same reasons it is not applicable in the offensive search algorithm. Note that deep pruning is possible when the intervening players are on their last branch [15].

In our experiments, we implemented all the pruning methods that are applicable for a given strategy. Paranoid can be reduced to a two-player game and full alpha–beta was used for it. For MaxN, we implemented immediate pruning and limited shallow pruning. For offensive we only implemented immediate pruning. When each of these strategies was used as part of the MP-mix algorithm (Algorithm 1), the relevant pruning techniques were used as well.

We now turn to present our main contribution—the MP-mix algorithm.

C. The MP-Mix Algorithm

The MP-mix algorithm is a high-level decision mechanism. When it is the player’s turn to move, he examines the situation and decides which propagation strategy to activate: MaxN, offensive, or paranoid. The chosen strategy is activated and the player takes his selected move.

The pseudocode for MP-mix is presented in Algorithm 1. It receives two numbers as input, \( T_d \) and \( T_o \), which denote defensive and offensive thresholds. First, it evaluates the evaluation values of each player via the evaluate() function. Next, it computes the leadingEdge, which is the evaluation difference between the two highest valued players and identifies the leading player (leader). If the root player is the leader and \( \text{leadingEdge} \geq T_d \), it activates the paranoid strategy (i.e., assuming that others will want to hurt him). If someone else is leading and \( \text{leadingEdge} \geq T_o \), it chooses to play the offensive strategy and attack the leader. Otherwise, the MaxN propagation strategy is selected. In any case, only one search from the leaves to the root will be conducted as the algorithm stops after the search is completed.

When computing the leadingEdge, the algorithm only considers the heuristic difference between the leader and the second player (and not the differences between all opponents). This difference provides the most important information about...
the game’s dynamics—a point where one leading player is too strong. To justify this, consider a situation where the leading edge between the first two players is rather small, but they both lead the other opponents by a large margin. This situation does not yet require explicit offensive moves towards one of the leaders, since they can still weaken each other in their own struggle for victory, while, at the same time, the weaker players can narrow the gap.

The implementation of the evaluate(i) function for the leading edge can vary. It can be exactly the same evaluation function that is being used in the main search algorithm, or any other function that can order the players with respect to their relative strength. A different function might be considered due to computational costs, or due to its accuracy.

D. Influence of Extreme Threshold Values on MP-Mix

The values $T_d$ and $T_o$ have a significant effect on the behavior of an MP-mix player (a player that uses the MP-mix framework). These values can be estimated using machine learning algorithms, expert knowledge, or simple trial-and-error procedures. Decreasing these thresholds will yield a player that is more sensitive to the game’s dynamics and reacts by changing its search strategy more often.

In addition, when setting $T_o = 0$ the player will always act offensively when he is not leading. When setting $T_d = 0$ the player will always play paranoid when leading. If both are set to 0 then the players always play paranoid when leading or offensive when not leading. When setting the thresholds to values that are higher than the maximal value of the heuristic function, we will get a pure MaxN player. Formally, let $G$ be a single-winner, $n$-players ($n > 2$) game, $T_v$, $T_d$ be the threshold values (we denote $T$ to refer to both) and $V$ a single vector of values at time $t$, where $v_i^t$ is the score value of player $i$ at time $t$. Assume that a player is using the MP-mix algorithm. Let $N(G, T)$ be the number of times that MP-mix will choose to execute the paranoid algorithm in a given run of the game. The following two extreme behaviors will occur.

Property 3.1 (MP-Mix on High $T$ Values): If for every time stamp $t$ and every player $i$, $v_i^t \leq T$ then $N(G, T) = 0$.

When setting the threshold too high (larger than the maximal possible value of the $v_i$), MP-mix behaves as a pure MaxN player, as no change in strategy will ever occur.

Property 3.2 (MP-Mix on Low $T$ Values): Let $x$ be the number of times leadingEdge $\geq 0$, then if $T = 0$, $N(G, T) = x$.

In the other extreme case, when the threshold is set to zero, a paranoid or offensive behavior will occur every time the MP-mix player leads (i.e., MaxN will never run). The above properties will come into play in our experimental section as we experiment with different threshold values that converge to the original algorithms at the two extreme values.

IV. EXPERIMENTAL RESULTS

In order to evaluate the performance of MP-mix, we implemented players that use MaxN, paranoid, and MP-mix algorithms in three popular games: the Hearts card game, Risk the strategic board game of world domination, and the Quoridor board game. These three games were chosen as they allow us to evaluate the algorithm in three different types of domains, and as such increase the robustness of the evaluation.

1) Hearts is a four-player, imperfect-information, deterministic card game.

2) Risk is a six-player, perfect-information, nondeterministic board game.

3) Quoridor is a four-player, perfect-information, deterministic board game.

In order to evaluate the MP-mix algorithm, we performed a series of experiments with different settings and environment variables. We used two methods to bound the search tree.

- Fixed depth. The first method was to perform a full width search up to a given depth. This provided a fair comparison to the logical behavior of the different strategies.

- Fixed number of nodes. The paranoid strategy can benefit from deep pruning while MaxN and offensive cannot. Therefore, to provide a fair comparison we fixed the number of nodes $N$ that can be visited, which will naturally allow the paranoid to enjoy its pruning advantage. To do this, we used iterative deepening to search for game trees as described by [8]. The player builds the search tree to increasingly larger depths, where at the end of each iteration he saves the current best move. During the iterations he keeps track of the number of nodes it visited, and if this number exceeds the node limit $N$, he immediately stops the search and reruns the current best move (which was found in the previous iteration).

A. Experiments Using Hearts

1) Game Description: Hearts is a multiplayer, imperfect-information, trick-taking card game designed to be played by exactly four players. A standard 52 card deck is used, with the cards in each suit ranking in decreasing order from Ace (highest) down to Two (lowest). At the beginning of a game the cards are distributed evenly between the players, face down. The game begins when the player holding the Two of clubs card starts the first trick. The next trick is started by the winner of the previous trick. The other players, in clockwise order, must play a card of the same suit that started the trick, if they have any. If they do not have a card of that suit, they may play any card. The player who played the highest card of the suit which started the trick, wins the trick.

Each player scores penalty points for some of the cards in the tricks they won, therefore players usually want to avoid taking tricks. Each heart card scores one point, and the queen of spades card scores 13 points. Tricks which contain points are called “painted” tricks. Each single round has 13 tricks and distributes 26 points among the players. Hearts is usually played as a tournament and the game does not end after the deck has been fully played. The game continues until one of the players has reached or exceeded 100 points (a predefined limit) at the conclusion of a trick. The player with the lowest score is declared the winner.

While there are no formal partnerships in Hearts, it is a very interesting domain due to the specific point-taking rules. When playing Hearts in a tournament, players might find that their best interest is to help each other and oppose the leader. For

2In our variation of the game we did not use the “shoot the moon” rule in order to simplify the heuristic construction process.
example, when one of the players is leading by a large margin, it will be in the best interest of his opponents to give him points, as it will decrease his advantage. Similarly, when there is a weak player whose point status is close to the tournament limit, his opponents might sacrifice by taking painted tricks themselves, as a way to assure that the tournament will not end (which keeps their hopes of winning). This internal structure of the game calls for the use of the MP-mix algorithm.

2) Experiments’ Design: We implemented a Hearts playing environment and experimented with the following players:

1) random (RND): this player selects the next move randomly from the set of allowable moves;
2) weak rational (WRT): this player picks the lowest possible card if he is starting or following a trick, and picks the highest card if it does not need to follow suit;
3) MaxN (MAXN): it runs the MaxN algorithm;
4) paranoid (PAR): it runs the paranoid algorithm;
5) MP-mix (MIX): it runs the MP-mix algorithm (thresholds are given as input).

In Hearts, players cannot view their opponents’ hands. In order to deal with the imperfect nature of the game, the algorithm uses a Monte Carlo sampling-based technique (adopted from [2]) with a uniform distribution function on the cards. It randomly simulates the opponents’ cards a large number of times (fixed to 1000 in our experiments), runs the search on each of the simulated hands, and selects a card to play. The card finally played is the one that was selected the most among all simulations. The sampling technique is crucial in order to avoid naive and erroneous plays, due to improbable card distribution.

When the players build the search tree, for each leaf node they use an evaluation function that uses a weighted combination of important features of the game. The evaluation function was manually tuned and contained the following features: the number of cards which will duck or take tricks, the number of points taken by the players, the current score in the tournament, the number of empty suits in the hand (higher is better), and the numeric sum of the playing hand (lower is better).

The MIX player uses the same heuristic function that the PAR and MAXN players use for the leaves’ evaluation process. However, in order to decrease the computation time, we computed the leadingEdge by simply summing the tournament and game scores. Without this simplification we would have had to run the Monte Carlo sampling to compute the function, as the original function contains features which are based on imperfect information (e.g., number of empty suits).

In addition to these three search-based players, we also implemented the WRT and RND players in order to estimate the players’ performances in a more realistic setting in which not all players are search-based players. The WRT player simulates the playing ability of a novice human player that is familiar solely with the basic strategy of the game, and the RND player is a complete newcomer to the game and is only familiar with the games’ rules, without any strategic know how. While these two players are somewhat simplistic players that are lacking the reasoning capabilities of the search-based players, their inclusion provided us with a richer set of benchmark opponents to evaluate the algorithm.

3) Results:

Experiment 1: Fixed Depth Bound $T_e = \infty, T_d \in [0, 50]$. Our intention in this experiment is to compare the performance of MIX with that of MAXN and PAR, and gain an understanding on the potential benefit of dynamically switching node propagation strategies. As such, in our first set of experiments, we fixed the strategies of three of the players and varied the fourth player. The first three players were arbitrarily fixed to always be (PAR, MAXN, and PAR) and this served as the environmental setup for the fourth player which was varied as follows. First, we used MIX as the fourth player and varied his defensive threshold $T_d$ from 0 to 50. To evaluate the advantages of a defensive play when leading, the offensive threshold $T_e$ was set to $\infty$. We then used MAXN and PAR players as the fourth player, in order to compare their performances to that of the MIX player in the same setting. The permutations table above shows the different permutations that were used.

We compared the behavior of the different settings of the fourth player. For each such setting we ran 800 tournaments, where the limit of the tournament points was set to 100 (each tournament usually ended after 7–13 games). The depth of the search was set to 6 and the technical advantage of paranoid (deep pruning) was thus neglected. The results in Fig. 6 show the difference in the tournaments’ winning percentages of the fourth player and the best player among the other three fixed players. A positive value means that the fourth player was the best player as it achieved the highest winning percentage, whereas a negative value means that it was not the player with the highest winning percentage.

The results show that PAR was the worst player (in this case a total of three PAR players participated in the experiment) resulting in around $−11\%$ winning less than the leader (which
in this case was the MAXN player). The other extreme case is presented in the rightmost bar, where the fourth player was a MAXN player. In this case, he lost by a margin of only 5% less than the winner. When setting the fourth player to a MIX player and the defensive threshold at 0 and 5, he still came in second. However, when the threshold values increased to 10 or higher, the MIX player managed to attain the highest winning percentage, which increased almost linearly with the threshold. The best performance was measured when \( T_d \) was set to 25. In this case, the MIX player significantly outperformed both MAXN and PAR players, as he attained a positive winning difference of 11% \( \{6 - (-5)\} \) or 17% \( \{6 - (-11)\} \), respectively \( (P < 0.05) \). Increasing the threshold above 50 will gradually decrease the performance of the MIX player, until it converges to the MAX player’s performance.

**Experiment 2: 50-K Nodes Search** \( T_o = 20, T_d = 20 \): In the second experiment, we decided to add to the pool of players the two extreme versions of MP-mix, denoted as OMIX and DMIX, in order to evaluate their performances as standalone players. OMIX is an offensive oriented MP-mix player with \( T_o = 20, T_d = \infty \), while DMIX is a defensive oriented MP-mix player with \( T_o = \infty, T_d = 20 \). The MIX player will be set with \( T_o = 20, T_d = 20 \). Overall, we used the following set of players \{MAXN, PAR, OMIX, DMIX, MIX\}. The environment was fixed with three players of the MAXN type and for the fourth player we plugged in each of the MP-mix players described above. In addition, we changed the fixed depth limitation to a 50-K node limit, so the paranoid search would be able to perform its deep pruning procedure and search deeper under the 50-K node limit constraint.

The results from running 500 tournaments for each MIX player are presented in Fig. 7. The best player was the MIX player that won over 32% of the tournaments, which is significantly better \( (P < 0.05) \) than the MAXN or PAR results. The DMIX came in second with 28%. The PAR player won slightly over 20% of the tournaments. Surprisingly, the OMIX player was the worst one, winning only 16% of the tournaments. The reason for this was that the OMIX player took offensive moves against three MAXN players. This was not the best option due to the fact that when he attacks the leading player he weakens his own score but at the same time the other players advance faster towards the winning state. Thus, in this situation, the OMIX player sacrifices himself for the benefit of the others. We assume that OMIX is probably better when other players are using the same strategy.

**B. Experiments Using Risk**

Our next experimental domain is a multilateral interaction in the form of the *Risk* board game.

1) **Game Description**: The game is a perfect-information strategy board game that incorporates probabilistic elements and strategic reasoning in various forms. The game is a sequential turn-based game for two to six players, which is played on a world map divided into 42 territories and six continents. Each player controls an army, and the goal is to conquer the world, which is equivalent to eliminating the other players. Each turn consists of three phases.

1) **Reinforcement Phase**: The player gets a new set of troops and places them into his territories. The number of bonus troops is \( \text{(number of owned territories/3) + continent bonuses + card bonus} \). A player gets a continent bonus for each continent he controls at the beginning of his turn, and card bonus gives additional troops for turning in sets. The card bonus works as follows: each card has a picture {cavalry, infantry, cannon} and a country name. At the end of each turn, if the player conquers at least one country, he draws a card from the main pile. Three cards with the same picture, or three cards with each of the possible pictures, can be turned in at this phase to get additional bonus troops.

2) **Attack Phase**: The player decides from which countries to attack an opponent’s country. The attack can be between any adjacent countries, but the attacker must have at least two troops in the attacking country; the battle’s outcome is decided by rolling dice. This phase ends when the player is no longer capable of attacking (i.e., he does not have any adjacent opponent country with more than two troops in it), or until he declares so (this phase can also end with zero attacks). After an attack is won the player selects how to divide the attacking force between the origin and source countries.

3) **Fortification Phase**: The player can move armies from one of his countries to an adjacent country which he owns. This rule has many variations on the number of troops one can move and on the allowable destination countries.

**Risk** is too complicated to formalize and solve using classical search methods. First, each turn has a different number of possible actions which changes during the turn, as the player can decide at any time to cease his attack or to continue if he has territory with at least two troops. Second, as shown in [5], the number of different opening moves for a six-player game is huge \( (\approx 3.3 \times 10^{38}) \) when compared to a classic bilateral board game (400 in *Chess* and 144 780 in *Go*). State-of-the-art search algorithms cannot provide any decent solution for a game of this complexity. Previous attempts to play *Risk* used either a heuristic-based multiagent architecture, where players control countries and bid for offensive and defensive moves [5], or a
might realize that unless they put explicit effort into attacking the leader, it will soon be impossible for them to prevent the leading player from winning. At the same time, the leading player might understand that everybody will turn against him, and decide to switch to a paranoid play, which might yield defensive moves to guard its borders. In case the situation changes again and this player is no longer a threat, he might switch his strategy again to his regular self-maximization strategy, namely MaxN.

2) Experiments’ Design: We worked with the Lux Delux\textsuperscript{4} environment that is a Java implementation of the Risk board game with an API for developing new players. We implemented three types of players: MAXN (using the MaxN algorithm), PAR (using the paranoid algorithm) and MIX (using the MP-mix algorithm).

Our evaluation function was based on the one described in [5], as it proved to be a very successful evaluation function that does not use expert knowledge about the strategic domain. In the reinforcement phase, it recursively computes a set of possible goals for each country, denoted as goal list, where its value is computed according to some fixed predefined formula (e.g., countries which control many borders have higher values than others). The next step was to get the highest offensive bid (i.e., the move with the most valuable goal list) and a defensive bid (i.e., the number of armies the country needed to acquire in order to be able to defend itself with a certain predefined probability) from each country, and distribute the armies according to the winning bids.

In the attack phase, the player attacks according to the winning offensive bids, as long as it exceeds some predefined winning probability. For example, assume that the goal list for the player that controls Congo is \{N. Africa, Brazil, Peru, Argentina\}. In this offensive bid the player attacks N. Africa, then Brazil, Peru, and Argentina. However, if during an attack the player sustains many casualties, resulting in a lower-than-threshold probability of completing its goal, it will decide to halt the attack and remain in its current position. The fortification phase also follows a similar simple auction protocol for the fortification of countries that have the highest need for defensive armies.

Experiment 1: Search-Based Agents $T_a = \infty, T_d \in [0, 40]$. In our first experiment, we ran environments containing six players, two players of each of the following types: MIX, MAXN, and PAR. The turn order was randomized (the playing order has less impact in the Risk game), and we used the “lux classic” map without bonus cards. In addition, the starting territories were selected at random and the initial placement of the troops in the starting territories was uniform. To avoid the need for simulating bidding phases, the leading edge function was simplified to consider only the current amount of troops and next round bonus troops.

Fig. 9 presents the results for this environment where we varied $T_d$ of the MIX players from 0 to 40. $T_a$ was fixed to $\infty$ in order to study the impact of defensive behavior and the best value for $T_d$. The numbers in the figure are the average winning percentage per player type for 750 games. The peak

\textsuperscript{4}http://sillysoft.net/lux/
performance of the MIX algorithm occurred with $T_d = 10$ where it won 43% of the games. We do not know exactly why the peak occurred at $T_d = 10$, but it is obviously a function of the heuristic that was used. A different function might have peaked at different value, if at all. By contrast PAR won 30% and MAXN won 27% (significance with $\mathcal{P} < 0.001$). The MIX player continued to be the leading player as the threshold increased to around 30. Obviously, above this threshold the performances converged to that of MAXN since the high thresholds almost never resulted in paranoid searches.

**Experiment 2: Specialized Players**

In the second experiment, we used three specialized expert knowledge players of different difficulty levels to create a varied environment. All three players were part of the basic Lux Delux game package: the *Angry* player was a player under the “easy” difficulty level, *Yakool* was considered “medium,” and *EvilPixie* was a “hard” player in terms of difficulty levels. These new players, together with the search-based players: PAR, MAXN, and MIX (with $T_d = 10$, $T_o = 10$) played a total of 750 games with the same environment setting as the first experiment (classic map, no card bonus and random, uniform starting position).

The results show that in this setting again, the MIX player achieved the best performance, winning 27% of the games, *EvilPixie* was runner-up winning 20% of the games, followed by the MAXN and PAR players winning 19% and 17%, respectively (significance with $\mathcal{P} < 0.001$). *Yakool* achieved 15% and *Angry* won 2%.

**C. Experiments Using Quoridor**

Following the above domains, with *Hearts* being an imperfect information game and *Risk* containing nondeterministic actions, we now move to evaluate the MP-mix algorithm in a perfect information and deterministic domain. Such a domain will provide a more explicit comparison of the MP-mix algorithm to the MaxN and to the paranoid algorithms. For that we selected the *Quoridor* board game as our third domain.

1) **Game Description:** *Quoridor*\(^5\) is a perfect information board game for two or four players, that is played on a $9 \times 9$ grid (see Fig. 11). In the four-player version, each player starts with five wall pieces and a single pawn that is located at the middle grid location on one of the four sides of the square board. The objective is to be the first player to reach any of the grid locations on the opposite side of the board. The players move in turn-wise sequential ordering, and at each turn, the player has to choose either to:

1) move his pawn horizontally or vertically to one of the neighboring squares;

2) place a wall piece on the board to facilitate his progress or to impede that of his opponent.

The walls occupy the width of two grid spaces and can be used to block pathways around the board as players cannot jump over them and must navigate around them. When placing a wall, an additional rule dictates that each player has to have at least one free path to a destination on the opposing side of the board. That prevents situations in which players team up to enclose a pawn inside four walls. Walls are a limited and useful resource and they cannot be moved or picked up after they have been placed on the game board.

*Quoridor* is an abstract strategic game that bears some similarities to *Chess* and *Checkers*. The state–space complexity in *Quoridor* is composed of the number of ways to place the pawns multiplied by the number of ways to place the walls, minus the number of illegal positions. Such estimation was computed in [3] for the two-player version of the game. In terms of size of the search space, the two-player version of the game is in between *Backgammon* and *Chess*. Obviously the search space increases dramatically when playing the four-player version of the game.

2) **Experiments’ Design:** We implemented a game environment in C++. The game board was constructed as a graph and Dijkstra’s algorithm was used to check the legality of wall positions (i.e., to check that there exists a path to the goal). We used a simple and straightforward heuristic evaluation function that sums the total distance of each of the players to the goal. Each player seeks to minimize his own distance while maximizing the opponents’ distances. In addition, to cope with the large branching factor of the game, we further limited the possible locations that a wall can be placed to a fixed radius around the pawns. We implemented the same search-based players: MIX,

\(^{5}\)More information on that game can be found in the creator’s website: http://www.gigamic.com/
MAXN, and PAR. We also implemented a somewhat “intelligent” RND player that picked the best move according to a randomly generated preferences vector that was created at the beginning of each game.

The experiments in this domain were very costly in computing hours as the branching factor was very large, around 64 (4 moves + 16 wall positions times 4 players, under the restricted radius-based wall placement rule). This was in contrast to the Risk experiments in which we artificially cut the branching factor to the set of most promising plans of attack. The experiments were run on a cluster of 32 multicore computers. To illustrate the required running time, a single depth five game with two search-based players and two nonsearch players could take between a few hours to two days to complete on a single central processing unit (CPU).

Experiment 1: Finding $T_o$ and $T_d$: In the first experiment on this domain, we started with looking for a good approximation for the threshold values. While in the previous domains we did some random exploration of these values, here we conducted a methodological brute-force search on all possible values. The first step was to run trial experiments to get an approximation of the maximum and minimum leading edge values of our heuristic function. We then discretized that range and ran a systematic search on all possible (discretized) values, where in each we played 500 games with MIX against three RND opponents. We ran the searches with the MIX player searching to depths 1, 2, and 3.

Fig. 12 presents an average of the results where at each $T_o, T_d$ combination, the $z$-axis presents the winning percentage for the MIX player playing against three RND opponents. From that surface we can see that the best observed values are in the neighborhood of $T_o = 4$ and $T_d = 7$. From this point on, all the reported experiments were conducted with the MIX player using these threshold values.

Experiment 2: MAXN Versus MIX Comparison: In the second set of experiments, we set up a comparative match-up between a MAXN and a MIX player. To complement these two search-based players we used two RND players. We ran 500 games at each different search depth and compared the amount of wins that each player attained.

The results for that experiment are depicted in Fig. 13, where it is easy to see that the MIX player, running the MP-mix algorithm with $T_o = 4$ and $T_d = 7$, achieved significantly better performances across all depth searches (significance with $P < 0.1$).

Experiment 3: PAR Versus MIX Comparison: We conducted a similar set of experiments where the paranoid algorithm (PAR) played against the MP-mix algorithm (MIX). Here again, to complement these two search-based players we used two RND players. We ran 500 games at each different search depth and compared the amount of wins that each player attained. However, as one of the main strengths of the paranoid algorithm is its ability to deep prune, we facilitated this advantage by allowing the paranoid algorithm to search one level deeper than the MP-mix algorithm (i.e., a search depth limit of one represents a depth one search for the MIX player, and a search depth of two for the PAR player).

The results for these experiments are depicted in Fig. 14, where it is easy to see that the MIX player, running the MP-mix algorithm with $T_o = 4$ and $T_d = 7$, achieved significantly better performances across all depth searches (significance with $P < 0.1$), even though paranoid searched one level deeper. The results show quite convincingly that the paranoid assumption is not a very successful approach in this specific domain. Moreover, similar results were observed when both algorithms used the same depth search.

Experiment 4: Search Players Versus 3 RND Opponents: These following experiments were conducted in order to evaluate and compare the individual performance of the different search algorithms against three RND opponents. Recall that an RND player is a player whose preferences were randomized to
reflect a wider range of social behavior than the simple rational, self-maximizing players, as is often observed in human playing strategies. Moreover, running against three RND players allows us to explore one depth deeper (that is to depth 6), and observe the performances at this depth.

The results are depicted in Fig. 15. Each column represents the winning percentage of the respective player against three RND players. We can see that the MIX player attains a higher winning percentage than the other two algorithms (except for depth 3, in which MAXNs with 45.2% were slightly higher than MIXs with 44.8%. However, these improvements do not appear to be as significant as in the direct match-up experiments, probably due to the limited competence of the opponents. We see a significant improvement in depth 6, where the MIX player managed to attain 51% winning, compared to the 37% and 35.2% of MAXN and PAR, respectively.

Experiment 5: Analyzing MP-Mix’s Behavior: Another interesting question is regarding the quantity of the different searches that the MP-mix algorithm performed. In other words, how do the amount of paranoid, MaxN, and offensive search procedures change with different search depths and opponents’ types. Figs. 16–18 show the percentages of each of the different node propagation strategies in the Quoridor game, for the experiments depicted in Figs. 13–15, respectively. These figures show a few interesting insights. First, it seems that the MaxN procedure was the most frequently used in all three settings, as it accounts for about 45% of the searches (with some variance to depth and opponents’ types). This seems intuitive as this is the “default” search procedure embedded in the meta-decision algorithm. Second, it also seems that beyond a search depth of 2, the remaining 55% are equally split between paranoid and offensive node propagation strategies. Finally, all the figures demonstrate a small, but monotonic increase in the percentage of paranoid searches with the increase in depth search. Naturally, these graphs will change with different heuristic function and threshold values. However, they do give us some sense of the amount of different searches that were conducted by the MP-mix agent in this domain.

V. OPPONENT IMPACT

The experimental results clearly indicate that MP-mix improved the players’ performances. However, we can see that the improvements in Risk and Quoridor were much more impressive than in the Hearts domain. As such, an emerging question is: under what conditions will the MP-mix algorithm be more effective and advantageous? To try and answer this question, we present the OI factor, which measures the impact that a single player has on the outcome of the other players. The OI factor is a meta property which characterizes a game according to its players’ ability to take actions that will reduce the evaluation of their opponents. This property is related both to the game and to the evaluation function that is used.

A. Intuition and Definition

The intuition behind the OI is that certain games were designed in a way that it is reasonably easy to impede the efforts of an opponent from winning the game, while in other games...
those possibilities are fairly limited. For instance, in Quoridor, as long as the player owns wall pieces, he can explicitly impede an opponent by placing them in his path. In contrast, in the game of Hearts, giving points to a specific opponent often requires the fulfillment of several preconditions, which are often attainable only through implicit cooperative strategic behavior among players.

Before presenting our proposed definition of the OI, let us first take a step back and discuss the notion of an evaluation function. In single agent search problems there is a clear and well-accepted definition for the semantic of an evaluation function, namely an estimation of the distance from the current state to goal state. However, when looking at adversarial search domains, the semantic of the evaluation function is not properly defined. It is widely accepted that the evaluation is a function from a game state to a number that represents the merit of the game state. Another example is to goal state. However, when looking at adversarial search domains, the semantic of the evaluation function is not properly defined. It is widely accepted that the evaluation is a function from a game state to a number that represents the merit of the

We will proceed by taking the most intuitive interpretation, which is also analogous to the interpretation used in the single agent search problems, and use a granularity-based model of the heuristic evaluation function

\[ H: \text{state} \to \{ e_1, e_2, \ldots, e_n \} \quad \forall i, e_i \in \{ \min, b, 2b, \ldots, \max \}, \]

where \( \max \) and \( \min \) are the maximum and minimum possible values of any single element in the evaluation vector and \( \delta \) represents the smallest nonnegative difference between consecutive values. With \( \max \) and \( \min \) fixed, finer grained evaluations can be achieved by reducing \( \delta \) and coarser grained evaluations can be achieved by increasing \( \delta \).

We are interested in describing games by the amount of change one player can inflict on the other. For that, we first define the notion of an influential state.

**Definition 5.1 (Influential State):** A game state for player \( A \) with respect to player \( B \) is called an influential state, if action \( \alpha \) exists such that the heuristic evaluation of \( B \) is reduced after activating \( \alpha \) by \( A \).

We can now define \( \text{InfluentialStates}(G, H) \) for a game \( G \) and a heuristic function \( H \) to be a function that returns the set of influential states with respect to any two players. \( \text{TotalStates}(G, H) \) will return the set of all game states.

**Definition 5.2 (Opponent Impact):** Let \( G \) be a game, \( H \) be a heuristic function, then

\[ \text{OI}(G, H) = \frac{|\text{InfluentialStates}(G, H)|}{|\text{TotalStates}(G, H)|}. \]

The OI factor of the game is defined as the percentage of influential states in the game with respect to all players. The intuition behind it can be illustrated with the following examples. Consider the popular game of Bingo, where each player holds a playing board filled with different numbers. Numbers are randomly selected one at a time and the first player to mark all the numbers in his playing board is the winner. It is easy to see that there is no way for one player to affect the heuristic score of another player. Thus, the OI of that game would be zero (as \( |\text{InfluentialStates}(G, H)| = 0 \)). Another example is the game of Yahtzee. The objective in Yahtzee is to score the most points by rolling five dice to certain combinations such as “three-of-a-kind” and “full house.” Once a combination has been used in the game, it cannot be used again. While there exist strategic considerations regarding the combinations assignments and dice rolling (e.g., play safe or risky), these decisions still lack any explicit impact on the opponent player.

Let us now consider the game of GoFish. In GoFish, the objective is to collect “books,” which are sets of four cards of the same rank, by asking other players for cards the player thinks they might have. The winner is the player who has collected the highest number of books when no cards are left in the players’ hands or in the deck. Here, theoretically, at any given state the player can decide to impact an opponent by asking him for a card. The opponent’s impact value of GoFish is equal to 1 (as \(|\text{InfluentialStates}(G, H)| = |\text{TotalStates}| \)).

In addition, there are games that can be divided into two parts with respect to their OI value. For example, in Backgammon, both players can usually hit the opponent’s pawns if they are open (“blot”), yielding a game with an OI larger than zero. However, the final stage of a game (called “the race”), when the opponent’s pawns have passed each other and have no further contact has an OI of zero. In Quoridor, the OI is very high as long as the player still owns wall pieces. After placing all the walls, his ability to impact the other players is reduced to almost zero.

The definition of OI is presented as a function of the game \( G \) and the heuristic evaluation function \( H \) that is used. While the OI of the game is obviously captured in the above examples, the role of the selected evaluation function is also an important one in determining the OI value. However, as the semantic of the evaluation function is not captured in its definition, one can use “unreasonable” (semantically wise) heuristic evaluation functions such as simple binary values (1 for a win, 0 otherwise), or a random function. While these functions will have different impact on the OI measure, our discussion will focus on functions that are semantically valid according to our granularity model.

**Property 5.1 [Convergence When OI(G, H) = 0]:** For game \( G \), in which \( \text{OI}(G, H) = 0 \), MaxN, paranoid, and MP-mix will behave exactly the same.

When the opponent’s level cannot impact the root player’s evaluation values (as there are no effective states), the algorithms will select the action with the highest evaluation for the root. Thus, when using the same tie-breaking rule, they will select the same action.

**Property 5.2 [Prevention When OI(G, H) = 1]:** For game \( G \), in which \( \text{OI}(G, H) = 1 \), it is possible to prevent a player from ever winning the game.

Property 5.2 states that in games in which the OI equals one, it is possible, theoretically, to prevent a specific player from winning the game. Following a \( \delta \) increment in the evaluation function of some player, another player will always have the ability to reduce it back to its previous state [as \( \text{OI}(G, H) = 1 \)]. To illustrate this property, let us look again at the GoFish game. In GoFish, a single player can always ask for the same card that his target player just got, and thus always prevent him from

\[ ^6\text{It is more accurate to say that the OI is approximately } 1, \text{ as there are cases at the end of the game, when there is only one set left to collect and players may end up with no cards on their hand.} \]
winning. An additional corollary is that if each player targets a different player to decrease his evaluation, the game will become infinitely long. To prevent such situations, new rules are sometimes added to the game, for example, a GoFish rule prohibiting asking twice for the same card, or in Chess the rule that prevents three recurring actions.

B. Estimating the Opponent Impact

When trying to intuitively estimate the OI values for our three experimental domains we can see the following. In Risk, one has a direct impact on the merit of other players when they share borders, as they can directly attack one another. Sharing a border is common since the game board is an undirected graph with 42 nodes (territories), each with at least two edges. In Hearts, a player’s ability to directly hurt a specific player is considerably limited and occurs only on rare occasions, for instance, when three cards are open on the table, the specific player has the highest valued card and he can choose between painting the trick or not. Moreover, since Hearts is an imperfect-information game, the player can only believe with a certain probability that his move will have an impact on its specific target. In Quoridor, a complete information, deterministic game, as long as the player still owns an unplaced wall piece, in most cases he can slow down any of the other opponents.

Computing the exact value of OI is impractical in games with a large (exponential) number of states. However, we can estimate it on a large sample of random states. In order to estimate the OI of Hearts, Risk, and Quoridor, we did the following. Before initiating a search to select the action to play, the player iterated over the other players as target opponents, and computed their evaluation function. We then counted the number of game states in which the root player’s action would result in more than a single heuristic value for one of the target opponents. For example, consider a game state in which the root player has five possible actions. If the root player’s actions would result in at least two different heuristic values for one of the target players, we would count this state as an influential state; otherwise, when all the available actions result in the same evaluation, we would count it as a noninfluential state. In Quoridor, the process can be simplified as it is equivalent to counting the number of turns in which walls are still available. In all domains, we ran 100 tournaments for different search depths, and computed the OI factor by counting the percentage of influential states. We limited our experiments to depth 6 as it was the maximum depth for the Quoridor and Risk games.

The results, depicted in Fig. 19, show that the OI for Hearts is very low when the search depth limit is lower than 4 (4% in depth 1, and 8% in depth 2). For larger depths limits the OI value monotonically increased but does not exceed 40%. The OIs for Risk and Quoridor are estimated around 85%, on average. From these results, we can conclude the following:

\[
\text{OI(Bingo/Yahtzee)} = 0 \times \\
\text{OI(Hearts)} = 0.35 \times \text{OI(Risk)} \approx 0.85 \times \text{OI(Quoridor)} \approx 0.85 \times \\
\text{OI(GoFish)} = 1.
\]

The fact that Risk and Quoridor have a higher OI factor is reflected in the experiment results, as the relative performance of MIX is much higher than in the Hearts domain. In both Risk and Quoridor players have a larger number of opportunities to act against the leading player than in Hearts. In Hearts, even after reasoning that there is a leading player that should be the main target for painted tricks, the number of states which one could choose to act against the leader are limited. When looking at expert human players in action, we can often observe how the weaker players try to tactically coordinate to subdue the leader. An interesting observation from the Hearts graph is the jump in OI between depths three and four, which can be explained by the fact that at depth four the trick is searched completely and the effect on the opponents is always taken into account.

An important question which still remains open after observing the results, concerns the relation between the characteristics of the game and the benefit which players gain when switching to a paranoid approach. Naturally, the improvement that a player will gain when using the MP-mix strategy depends on his evaluation function and the threshold value. For instance, we can easily find an evaluation function in which the paranoid and the MaxN approaches will always yield the same result (e.g., simply pick a function which returns 1 for winning states and 0 otherwise), or, as seen above, we can set the threshold values to those which result in no changes in the algorithm at all.

VI. Conclusion

Generalizing adversarial search algorithms to more than two players is a challenging task. The MaxN and paranoid algorithms are two contrasting approaches, where the first assumes that each player maximizes its own heuristic value and the second assumes that each player tries to minimize the root player’s heuristic value.

Based on observations of human players’ behaviors in single-winner multiplayer games, we presented the MP-mix algorithm which interleaves both algorithms by running the paranoid algorithm in situations where it is leading the game by some predefined threshold, the directed offensive algorithm when it is being led by some predefined threshold, and otherwise the MaxN algorithm. The MP-mix algorithm dynamically changes the assumption about the opponents’ behavior, thus it overcomes a basic
shortcoming of the other algorithms in which this assumption remains fixed for the duration of the game.

We presented experiments in three domains with different properties: 1) Hearts—an imperfect-information, deterministic game; 2) Risk—perfect-information, nondeterministic game; and 3) Quoridor—perfect information, deterministic game. Our experiments demonstrated the advantages of using the MP-mix algorithm. In Hearts, the MP-mix algorithm outperformed the paranoid and MaxN algorithms in both fixed and random environments. The same results were achieved in Risk, where it significantly outperformed the other algorithms by a larger margin. The results in Quoridor show significant improvement when the MP-mix is facing the paranoid or the MaxN algorithm in four-player games.

Moreover, our results suggest that the genuine benefit in using the MP-mix algorithm in the Risk and Quoridor domains is much higher than in the Hearts domain. The proposed reason for this difference is related to a game characteristic, which we defined as the OI. The OI reflects the ability of a single player to explicitly impact the heuristic value of a specific opponent. In games with low OI values, the impact of playing the paranoid approach is weaker as there are fewer game states in which an opponent coalition can directly affect the root player.

VII. FUTURE DIRECTIONS

In terms of future research it would be interesting to apply machine learning techniques to learn the optimal threshold values for different leading edge functions. These values should be related not only to specific games, but also to the heuristic function which the players use. In addition, more research should be done in order to thoroughly understand the influence of the OI factor on the performance of MP-mix in different games. One might also consider generalizing the algorithm for games in which there is more than one winner. In games with a fixed number of winners the computation of the leading edge difference might be different. Further in the future it might also be interesting to look at the impact of dynamic node propagation strategies in partial-information games or nonconstant-sum games.

More research is also needed to evaluate the impact of the offensive strategy. Specifically, it will be interesting to study and compare the suggested strategy that prepares for the worst case by assuming that the opponents will not try to attack the leader, to a more optimistic strategy that assumes everyone will be serving in a coalition against the leader. The latter interpretation of the offensive strategy can be regarded as complementing the paranoid assumption. In addition, it might be worth examining an offensive strategy that does take (to some extent) its own evaluation into account.

Another open question that might be interesting to address is to apply the MP-mix decision procedure at every node of the search tree, as opposed to only before conducting the search. This will allow a more flexible behavior and potentially increase the accuracy of the search. We would also like to consider the integration of this approach with Monte-Carlo-based algorithms (e.g., the UCT algorithm [7]) that are gaining rapid popularity. Finally, it will be interesting to provide a comprehensive classification of various multiplayer games according to their OI value. This can be achieved via combinatorial study of the game states, or through extensive experimental work to count the number of states in which one player can affect the other.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their fantastic remarks. They would also like to thank Prof. S. Kraus for helping in earlier versions of this manuscript.

REFERENCES

Inon Zuckerman received the B.A. degree in computer science from the Interdisciplinary Center, Herzliya, Israel, in 2004, the M.S. degree in artificial intelligence from Katholieke Universiteit Leuven (K.U.Leuven), Leuven, Belgium, in 2005, and the Ph.D. degree in computer sciences from Bar-Ilan University, Tel Aviv, Israel, in 2009. He is a faculty member (Lecturer) in the Industrial Engineering and Management Department, Ariel University Center of Samaria, Ariel, Israel. Previously he worked as a Research Associate in the Institute for Advanced Computer Studies (UMIACS), University of Maryland at College Park. He works in the field of artificial intelligence, specifically (but not exclusively) on problems relating to the development of autonomous agents and multiagent systems, automated game playing, and evolutionary game theory.

Ariel Felner received the B.Sc and M.Sc. degrees in mathematics and computer science from the Hebrew University, Jerusalem, Israel, in 1993 and 1995, respectively, and the Ph.D. degree from Bar-Ilan University, Tel Aviv, Israel, in 2002. Since 2004, he has been a faculty member (currently Senior Lecturer) in the Department of Information System Engineering, Ben-Gurion University, Be’er-Sheva, Israel. His main research area is solving hard combinational problems in artificial intelligence such as games, puzzles, and map navigation with heuristic search. His research focuses on different heuristic search methods and applying them to different environments and circumstances.