Guiding Multiplayer MCTS by Focusing on Yourself

Hendrik Baier
Intelligent and Autonomous Systems Group
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
hendrik.baier@cwi.nl

Michael Kaisers
Intelligent and Autonomous Systems Group
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
michael.kaisers@cwi.nl

***Abstract*—In n-player sequential move games, the second root-player move appears at tree depth $n + 1$. Depending on n and time, tree search techniques can struggle to expand the game tree deeply enough to find multiple-move plans of the root player, which is often more important for strategic play than considering every possible opponent move in between. The minimax-based *Paranoid search* and *BRS+* algorithms currently achieve state-of-the-art performance, especially at short time settings, by using a generally incorrect opponent model. This simplifying model enables Alpha-Beta pruning, thus allowing the search to reach follow-up root player moves at greater search depths. This paper introduces *abstraction over opponent moves* to MCTS in multiplayer games, and uses its synergies with *progressive widening* in order to outperform these state-of-the-art minimax-type baselines. Progressive widening makes the search tree selective and deep enough to reach the root player's next moves, and abstraction over opponent moves generalizes value estimates of the root player's moves online across different opponent moves. In contrast to paranoid search approaches, opponent models do not have to be simplified. Experiments show that combining progressive widening with opponent move abstraction (MCTS-OMA-PW) leads to improved performance in the multiplayer games Chinese Checkers, Rolit, and Focus. Our work thus paves the way for improved multiplayer search by online generalisation that focuses on the root player's actions, with the potential of improving real-time MCTS applications as well as training in expert iteration and other meta-algorithms where short time settings are relevant.**

***Index Terms*—Game AI, Monte Carlo Tree Search, Multi-agent systems, Multiplayer games, Monte Carlo methods**

I. Introduction

Monte Carlo Tree Search (MCTS) and its many variants have celebrated countless successes in recent years, e.g. in General Game Playing [1], General Video Game Playing [2], and as an essential part in recent breakthroughs in deterministic two-player board games [3], [4]. When considering the exponential complexity of multiplayer games however, and in particular when only short search times are available, MCTS can struggle against alternative minimax-based search algorithms. MCTS has for example been found to be inferior to *Paranoid search* [5] and *Best-reply Search* (BRS) [6] in five out of eight tested multiplayer domains when compared at 250ms per move [7]. Time settings of much less than one second per search have several important applications however, such as real-time games, or the use of MCTS as a policy improvement operator within frameworks such as Expert Iteration [8] or Zero learning [4].

What makes Paranoid and BRS stronger in these cases? These algorithms make use of the *paranoid assumption* – the assumption that all opponent players are always conspiring against the root player. While often wildly incorrect, this simplified opponent model allows for an n-player game to be treated as a 2-player zero-sum game. This means that Alpha-Beta pruning can be applied, and an iterative deepening search can reach the next moves of the root player (at depths $n + 1$, $2n+1$ etc.) within a shorter search time. Reaching future moves of the root player, or at the very least the second root player move at depth $n + 1$, can often be a much more important factor for the strength of the resulting policy than considering many of the opponent moves in between, or modelling the preferences of these opponents correctly. We believe this to be the deciding factor in many multiplayer domains: the ability to focus *mostly on yourself* and your (at least short-term) plans, without getting too distracted by the vast number of potential opponent moves in between. In many domains, your own plans will be relatively robust to these opponent moves on average. However, an algorithm should be able to detect and adapt when an opponent does have the ability to interfere, as is to be expected in multiplayer games; simply ignoring opponents is generally not sufficient.

In this paper, we tackle this problem of multiplayer MCTS with a combination of the existing technique of *progressive widening* (PW) [9], [10] and the newly proposed technique of online generalization through *opponent move abstraction* (OMA). Progressive widening makes sure to grow the tree deeply enough to see follow-up root player moves, by initially focusing only on the most likely opponent moves in between (using offline generalization). Opponent move abstraction makes better use of value estimates learned for these follow-up moves, by generalizing them during search to similar game states that only differ in past moves of the opponents, but not in past moves of the root player (using online generalization). Both techniques have parameters that allow to specify exactly how many opponent moves we can afford to prune, and how strongly we can rely on the online generalization over different opponent moves. In the limit, they converge to no pruning and no generalization, and therefore do not change the asymptotic

978-1-7281-4533-4/20/$31.00

behavior of MCTS, while providing a significant boost at limited search times.

This paper is structured as follows: Section II discusses related work, including the minimax-type baselines, and the existing element of our multiplayer MCTS approach: progressive widening. Section III proposes the new element of our approach, opponent move abstraction, and argues for its synergy with progressive widening. Section IV outlines the six test domains, and Section V presents our experimental results. Finally, Section VI concludes and discusses future work.

II. Background

This section discusses the baselines Paranoid search and BRS+[1]; outlines the MCTS variant our approach is extending, and the *progressive widening* technique as one element of our approach; and mentions previous techniques for online generalization in MCTS in order to situate the newly proposed element of *opponent move abstraction* in the literature.

A. Paranoid search

Following notation from related work on BRS [12], we model the deterministic and fully observable games used in this work as tuples $(N, S, Z, A, T, P, u_i, h_i, s_0)$, where $N = \{1, \ldots, n\}$ is the set of players; S is the set of game states; $Z \subseteq S$ is the set of terminal states; A is the set of moves; $T : S \times A \to S$ is the transition function mapping any state s and any move chosen from the available moves $A(s) \subseteq A$ to the successor state resulting from taking the move; $P : S \to N$ maps any state to the player to move; $u_i : Z \to [0, 1] \subseteq \mathbb{R}$ returns the utility of a terminal state for Player i; $h_i : S \to [0, 1]$ returns the heuristic evaluation of a state for Player i (the result of a *static board evaluation function* like those discussed in Section IV); and $s_0 \in S$ is the starting state of the game. The tuple $\mathbf{u}(s) = (u_1(s), \ldots, u_n(s))$ refers to the utility of state s from the point of view of all players; the same holds for $\mathbf{h}(s)$.

Paranoid search [5] approaches multiplayer games by assuming that all opponent players have formed a coalition against the root player r. This effectively turns any multiplayer game into a two-player zero-sum game between r and the coalition, which makes Alpha-Beta style pruning possible. The utility $V_d(s, a)$ of any move a in any state s with a desired lookahead of depth d is defined as follows for Paranoid:

$$V_d(s,a) = \begin{cases} \mathbf{u}(s') & \text{if } s' \in Z \\ \mathbf{h}(s') & \text{if } s' \notin Z \text{ and } d = 0 \\ \max^r_{a' \in A(s')} V_{d-1}(T(s', a'), a') & \\ & \text{if } s' \notin Z \text{ and } d > 0 \text{ and } P(s) = r \\ \min^r_{a' \in A(s')} V_{d-1}(T(s', a'), a') & \\ & \text{if } s' \notin Z \text{ and } d > 0 \text{ and } P(s) \neq r \end{cases} \quad (1)$$

where $\max^r$ returns a tuple that maximizes the r^{th} value (here over all legal next moves, the utility tuple that maximizes the utility of the root player), $\min^i$ analogously, and $s' = T(s, a)$. Paranoid tends to perform well in practice due to the deeper searches made possible by Alpha-Beta style pruning. However, it suffers from using an overly pessimistic opponent model, which becomes more unrealistic the deeper it searches, and the more opponents it is facing. In many games it is ultimately not possible to win if all opponents form a coalition.

B. BRS and BRS+

Best-Reply Search (BRS) and BRS+ are improved search techniques based on the paranoid assumption. Like Paranoid, BRS assumes that all opponents are playing against the root player; however, it restricts the move choices of this coalition. In vanilla BRS [6], as used in prior comparisons to MCTS [7], only the opponent with the strongest move against the root player can act, while all other opponents have to pass. This means that all opponent levels of the tree are virtually combined into one, and the root player and the opponent coalition move alternatingly as if playing a two-player game.

The restricted move choice of opponents in BRS fulfills two functions. First, it allows for even deeper search and more long-term planning than in vanilla Paranoid search, and Alpha-Beta pruning can also still be applied. Second, it weakens some of the negative effects of the paranoid assumption: All opponents are still playing against the root player, but their moves and therefore their strength as a coalition are limited.

BRS has been shown to work well in a variety of multiplayer games [6], [13], [14]. However, it can lead to illegal game states in the tree when the game at hand does not actually allow passing. This motivated the further development of BRS into BRS+ [12]. BRS+ assumes access to a *static move ordering function*. Its basic idea is that whenever an opponent does not have the strongest move against the root player in a given state and would be forced to pass by vanilla BRS, this opponent instead gets to play the move ranked highest by the move ordering, which maintains valid game states during search. Subfigures 1(a) and (b) show the effect of this pruning – 1(a) is a toy example of a game tree, and 1(b) is the corresponding BRS+ search tree in which at most one of the opponents can make a free move (not ranked first by the move ordering) between two turns of the root player. The highest ranked moves are marked with bold lines in the illustration.

BRS+ and its generalization OPPS [11] can be considered state of the art for fully observable, deterministic multiplayer games[2]. Problems such as illegal moves are avoided, for the price of requiring a move ordering function, and being somewhat dependent on its quality.

C. MCTS and progressive widening

Monte Carlo Tree Search (MCTS) denotes a large family of related search algorithms [15]. We assume that the reader is familiar with its basic structure. In this work, we use the popular and simple *Upper Confidence Bounds for Trees* (UCT)

[1]Given the similarity in background setting, parts of these subsections are shared with a paper accepted for publication [11].

[2]Due to space restrictions, we only compare to BRS+ in this paper; OPPS is playing somewhat stronger but otherwise shows qualitatively similar results.

variant of MCTS [16], i.e. moves in the search tree are selected to maximize:

$$\bar{X}_a + k\sqrt{\frac{\ln n}{n_a}} \tag{2}$$

where $\bar{X}_a$ is our current value estimate for taking move a in the current node (the average reward for taking a here), n is the number of times we have visited the current node so far, n_a is the number of times we have selected move a in the current node so far, and k is the *exploration factor*, trading off exploration and exploitation. When an MCTS simulation leaves the tree, we use the same static evaluation functions as used by our minimax baselines to evaluate the newly visited state, add a node representing it to the tree, and then backpropagate its evaluation; i.e. we do not use rollouts.

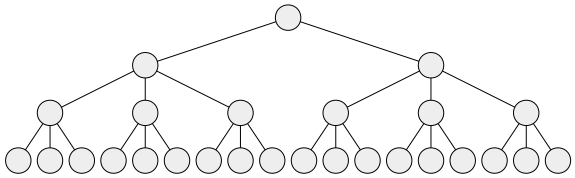
Progressive widening (PW) or progressive unpruning [9], [10], is a soft move pruning technique that allows the search tree to initially focus on moves that are heuristically believed to be promising, drastically reducing the branching factor of the tree and thus deepening the search. Over time, the technique adds more and more moves to the tree, in decreasing quality as ranked by a static move ordering function. This enables it to retain the asymptotic behavior of MCTS, i.e. the policy found as the number of MCTS simulations goes to infinity. Concretely, we determine the number of top-ranked moves considered in any given tree node by:

$$\lceil cn^{\alpha} \rceil \tag{3}$$

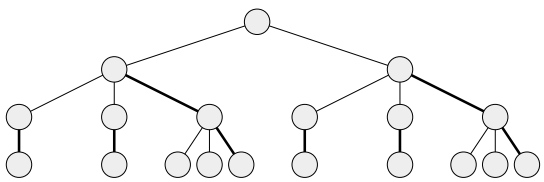
where n is the number of visits to the current node so far, and the widening factor $c > 0$ and widening exponent $\alpha \in]0, 1[$ are parameters determining the rate of unpruning over time.

D. Online generalization in MCTS

Recent efforts in game AI have largely focused on *offline* generalization, for example with the deep convolutional neural networks used by AlphaGo and AlphaZero [3], [4] for state value and policy estimation. Training these estimators offline means that they have to generalize across *all* states that could ever be visited in any match, and they therefore require not only very powerful function approximators, but also very time-intensive training.



(a) The fully expanded game tree of a three-player toy game.



(b) The pruned tree as searched by BRS+.

Fig. 1: Illustration of BRS+.

An alternative (and orthogonal) approach to learning in search is *online* generalization. The goal of online generalization is to focus learning on the states actually visited by the search in the currently ongoing match, and make the most out of the limited amount of data we can collect during play. The MCTS tree in its basic form estimates the value of each move a_d at search depth d in the precise context of its entire history $a_0 a_1 \dots a_{d-1}$, from the move a_0 made at the root to the preceding move a_{d-1}[3]. These value estimates are unbiased, but do not generalize or share any information across the tree. Move value estimation techniques that do generalize online may take the form of *abstractors*: They create a wider learning context by e.g. explicitly focusing on only a part of a move's history and generalizing over the rest, or abstracting the rest away. The resulting value estimates are biased, but can be learned much more quickly due to wider contexts appearing more often. Therefore, such abstract value estimates have a lower variance. They can then be used to guide the tree search into directions that appear promising based on past experience in similar situations, i.e. in the same context.

The Move-Average Sampling Technique (MAST) for example, proposed for General Game Playing [17], uses the widest possible learning context and estimates move values globally and independently of where the move is found in the tree. It learns how good a move is in general. The N-Gram Selection Technique (NST) [18], as another example, additionally learns move value estimates in the context of the immediately preceding move a_{d-1} (or the preceding sequence of two moves $a_{d-2}a_{d-1}$). It learns how good a move is as answer to (an)other move(s). For the Predicate-Average Sampling Technique (PAST) [17], the learning contexts are given by the predicates used in General Game Playing to describe states, and thus depend on the specifics of game states reached by certain histories. The Rapid Action Value Estimate (RAVE) technique [19] and its variants estimate the value of a move a_d in the context of every single prefix of its history: a_0, a_0a_1, $a_0a_1a_2$, …, $a_0 \dots a_{d-1}$– resulting in one move a_d leading to many value estimate updates, for every partial history corresponding to every move that was traversed before it.

In this work, we make use of an online generalization technique that abstracts over opponent moves in every move's history. The following section describes how these abstract value estimates are computed, and how they are used to influence move selection in the MCTS tree.

III. Opponent Move Abstraction

When studying the behavior of Paranoid and BRS+, we made an interesting observation regarding such approaches based on the paranoid assumption. When you let them play with increasing search depths, these algorithms do not necessarily get stronger with each additional tree level they reach, which would be generally expected from algorithms without the paranoid

[3]We are ignoring transposition tables here for clarity.

assumption (such as Alpha-Beta search in two-player zero-sum games, and max^n [20] in multiplayer games[4]). They instead tend to only significantly increase in strength whenever they reach an additional tree level *where the root player moves*.[5]

This makes intuitive sense: As the paranoid assumption about the opponents' objectives is usually incorrect, the behavior of opponents cannot be particularly well predicted by an algorithm using it, and searching an additional level of incorrect predictions does not help. However, paranoid algorithms still model their *own* objectives correctly, and the Alpha-Beta pruning enabled by the paranoid assumption allows them to search deeper and reach more of their *own* future moves. This suggests that *focusing on yourself* is a lot more important in multiplayer games than focusing on what opponents are doing. The improved strength of BRS and BRS+ compared to Paranoid confirms this as well, as these algorithms additionally use pruning of opponent moves in order to search even deeper trees that focus even more on the moves of the root player. Of course this does generally not mean that you can ignore your opponents completely or model them with static, non-branching policies; occasionally, with a frequency depending on the game at hand, managing complex interactions with opponents is crucial. But on average, in a game with several or many players, your plans are somewhat robust to specific move choices of specific opponents – because typically, not every opponent can strongly interfere with you at every turn.

The main goal of this paper is to apply this insight to the MCTS framework, in a way that makes use of its particular strengths. Our approach uses two elements and identifies a particular synergy between them: the existing technique of progressive widening as outlined above, and the newly proposed technique of opponent move abstraction (OMA) explained in the following. Two questions need to be answered.

First question: How are OMA value estimates computed? Opponent move abstraction keeps value estimates for all moves based on a context that abstracts over (i.e. ignores) past *opponent* moves, only including past moves of the *currently moving* player. Consider the example in Fig. 2, showing a depth-four toy tree of a three-player game between the root player and two opponents. The MCTS tree itself keeps separate and independent value estimates for the three occurrences of the root player move a, because they appear in different parts of the tree and therefore in potentially different situations, where they could have arbitrarily different values. They have different histories: xmo vs. xmp vs. ynq. OMA however maps the move a from node 1 and the move a from node 2 to the same abstract value estimate – because OMA abstracts over past opponent moves, both moves simply have the OMA context x. This means that when move a has for example been chosen 14 times from node 1, and 23 times from node 2, the OMA value estimate for the move a in the opponent-abstracting context x has already seen 37 updates. Note that move a when chosen from node 3 has a different OMA context (y) and therefore does not share the same OMA value estimate.

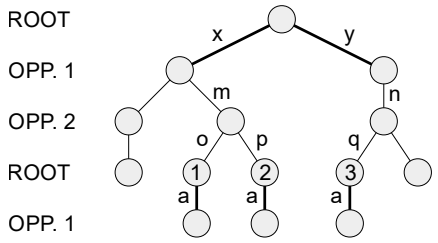


Fig. 2: Opponent Move Abstraction. The root player's OMA identifies the moves a from nodes 1 and 2, because they have the same history of root player moves, x. The move a from move 3 has a separate OMA value estimate.

Second question: How are OMA value estimates used? In every move selection in the search tree, we can now replace the maximization of UCT values as given by Equation 2 with maximizing a combination of the UCT value and the newly computed OMA value estimate. There are several ways to combine UCT values with additional value estimates from an online generalization approach like OMA, with different degrees of theoretical well-foundedness and complexity. For RAVE, for example, different suggestions were made in [19], [21], [22] – the performance in practice is often comparable. For this work, we are therefore choosing the following simple, heuristic formula [19]:

$$\beta \bar{X}_a^{OMA} + (1-\beta)\bar{X}_a + k\sqrt{\frac{\ln n}{n_a}} \tag{4}$$

where $\bar{X}_a^{OMA}$ is the OMA value estimate for move a in the current node, and β is the weighting coefficient as given by:

$$\sqrt{\frac{e}{3n_a + e}} \tag{5}$$

where e is a tunable "equivalence parameter" regulating the strength of OMA's influence. Decaying β ensures that OMA's low-variance, but potentially high-bias value estimate is kickstarting the search, but gradually phases out as the node collects more and more samples for its unbiased UCT estimate.

Our proposed approach to multiplayer search combines opponent move abstraction with progressive widening. In this setup, PW has the task of growing a tree deep enough to reach the root player's next moves, and OMA has the task of learning the value of these moves abstracting over all potentially irrelevant opponent moves in between. In this way, the crucial knowledge about your own future actions can be more easily acquired *and* more effectively used by MCTS.

Both elements of our approach, PW and OMA, have the additional advantage of converging in effect to zero as the search time goes to infinity: PW by widening to all legal moves in all tree nodes, and OMA by converging to an abstractor weight of zero. While we are considering short time settings here, tuning how fast this convergence happens is still valuable

[4]We are not describing max^n in more detail here, as it is usually weaker than Paranoid, and is therefore not used as baseline in our experiments.

[5]For example, BRS+ depth d against $d-1$ tested for $d = \{2, \ldots, 7\}$ won $56.5\%, 51.6\%, 48.0\%, \mathbf{89.3}\%, 40.1\%, 42.9\%$ in 4-player Chinese Checkers (of 500 games, 2$^{\text{nd}}$ root player move depth in bold).

for achieving optimal performance. Furthermore, this means that MCTS still converges to a Nash equilibrium in the limit in our setting [23]. Our minimax-type baseline algorithms do not have this convergence guarantee due to their unrealistic paranoid assumption, as well as due to the hard pruning of moves in BRS+.

Please note that while this explanation focuses on the root player, where the proposed combination of techniques has the most impact, both progressive widening and opponent move abstraction are applied at *all* nodes in the tree, not just those corresponding to the root player. Future work may expound the effect of treating the root player nodes and the opponent nodes separately. In the following, we empirically test our approach, and demonstrate the synergy of opponent move abstraction and progressive widening.

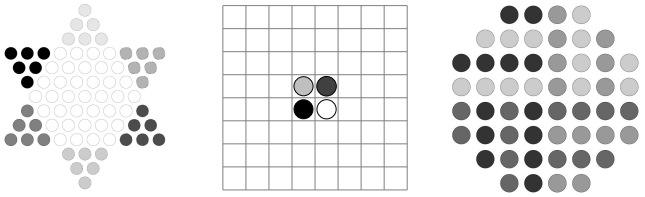
IV. Domains

We use six different test domains for the experiments in the following section: *Chinese Checkers* with three, four, and six players; *Rolit* with three and four players; and *Focus* with four players. All are deterministic, zero-sum, perfect-information, turn-taking board games, although none of these properties should in principle be limitations of our approach. These domains have been used in prior work on minimax-based and MCTS-based search algorithms, including a direct comparison of these two families of search algorithms [7], which partly inspired this work. We are staying close to the domains, evaluation functions and move ordering functions described in [7], and encourage the interested reader to consult the complete game rules and more detailed explanations therein.

Chinese Checkers is a game for two to six players, played on a star-shaped board, with every player trying to win by reaching the opposite corner of the board first. Jumping over own and opponent pieces is allowed. In order to speed up experiments, we follow previous work [7], [23] in using a small board with 73 fields and 6 pieces per player.

Both our MCTS-based approach as well as the minimax-based baselines require a static board evaluation function, used to assign heuristic values to leaf positions, as well as a static move ordering function, used to search more promising moves first, or pruning less promising moves. For Chinese Checkers, we use a slightly simplified version of the evaluation function from [7], which adds up for each player the distances of each piece to the far end of the goal corner, and then normalizes those values so they add up to 1 over all players. The move ordering is identical to the one used in [7], which orders all legal moves by how much closer they bring the moved piece to the far end of the goal corner.

Rolit is a game for three to four players, generalizing the two-player game of Othello. The players alternatingly place pieces on a 8×8 board, trying to capture opponent pieces in straight lines in between their own pieces. Captured discs take on the color of the capturing player, and when the board is filled, the player with the most pieces wins. For Rolit, we use a slightly tuned version of the evaluation function from [7], adding up for each player the number of *stable pieces* – pieces that cannot be captured anymore – and the number of legal moves on the current game board. The number of stable pieces is multiplied with a factor of two. The move ordering used here is identical to the one used in [7] and based on a static square-value table.



(a) Chinese Checkers with six players. (b) Rolit with four players. (c) Focus with four players.

Fig. 3: Starting positions of three of our test domains.

Focus is a game for two to four players, played on an 8×8 board as well, but with three squares removed from each corner. In Focus, pieces can be stacked on top of each other, and the entire stack or any number of its top pieces can then be controlled by the player on top. Stacks of size x can move x squares. When a stack exceeds five pieces, the bottom pieces are captured by the moving player such that the maximal stack size remains five, and a player wins either by being the only one with any legal moves left, or by capturing a given number of pieces (dependent on the number of players). For Focus, we use a slightly tuned version of the evaluation function from [7] as well, calculated for each player with the formula $300 - 50u + v$; u is the minimum number of captures the player still needs to win the game, and v is the number of stacks the player controls. The move ordering we use is identical to the one in [7] and gives a preference to moves creating larger stacks, as well as to moves that increase the number of stacks the player controls.

Fig. 3 shows the starting positions for six-player Chinese Checkers, four-player Rolit, and four-player Focus, adapted from [7].

V. Experimental Results

All experiments limit all players to 250ms of thinking time per move, as our work is mainly motivated by the weaker performance of multiplayer MCTS at short search times. Vanilla MCTS, i.e. UCT in this work, has one parameter: the exploration factor k. MCTS with progressive widening has two additional parameters: the widening exponent α and the widening factor c. MCTS with opponent move abstraction has two additional parameters as well: the equivalence parameter of the abstractor e, and a boolean f that determines whether abstracted value estimates should be forgotten in between moves, or accumulated throughout entire games. The parameters of all algorithm variants were first tuned against the vanilla MCTS baseline as well as against each other, followed by a final test of the best found parameter settings with 1000 games. The results of these final tests are presented here. Whiskers on bar plots indicate 95% confidence intervals. Numbers in

parentheses behind the names of games refer to their number of players, e.g. Rolit(4) is Rolit with four players. Minimax-based and MCTS-based algorithms use the same heuristic evaluation functions. MCTS uses no enhancements other than the ones described in this paper, and minimax uses no other enhancements than iterative deepening and the same move ordering that progressive widening uses.

Our experiments are divided into two sets. In the first set, described in Subsection V-A, we test the effect of adding opponent move abstraction to vanilla MCTS, as well as the effect of adding it to MCTS already using progressive widening. This serves to demonstrate the synergy effect between the two techniques. In the second set, presented in Subsection V-B, we then test the minimax-based baselines Paranoid and BRS+ against vanilla MCTS, MCTS with opponent move abstraction (MCTS-OMA), MCTS with progressive widening (MCTS-PW), and against MCTS with both techniques (MCTS-OMA-PW). This is to demonstrate the improvement achieved over the state of the art for such short multiplayer searches.

A. Combining OMA and PW in MCTS

Our first set of experiments compares the performance of opponent move abstraction when added to vanilla MCTS, shown in Fig. 4, to the performance when added to MCTS already equipped with progressive widening, shown in Fig. 5. OMA leads to a modest improvement in vanilla MCTS at these short time controls; the improvement is statistically significant at the 95% confidence level in all domains except for 3-player Rolit. The win rate of MCTS-OMA vs. MCTS, averaged over all tested games, is 57.3%.

For MCTS with progressive widening however, OMA leads to significant improvements in all tested domains, with an average win rate of MCTS-OMA-PW vs. MCTS-PW of 65.0%. The improvement from adding OMA when progressive widening is present is significantly larger than the improvement from adding OMA when it is not present in all domains except for 4-player Focus. This synergy effect is caused by PW helping the search to grow a tree deep enough to reach more follow-up root player moves, whose value estimates can then be generalized by OMA and enhance the rest of the search process. This makes the two techniques especially well-suited for use in combination.

B. MCTS with PW and OMA vs. paranoid and BRS+

In our second set of experiments, we compared all combinations of progressive widening and opponent move abstraction added to MCTS – vanilla MCTS, MCTS-OMA, MCTS-PW, and MCTS-OMA-PW – against the minimax-based baselines Paranoid and BRS+, representing the state of the art for such short multiplayer searches.

Fig. 6 shows the performance of vanilla MCTS against the baselines. In seven out of twelve comparisons, MCTS is the weaker player, with an average win rate across all games of only 46.0% against Paranoid and 34.5% against BRS+. This confirms that at least for these basic forms of the algorithms, minimax-based searching is superior under our test conditions.

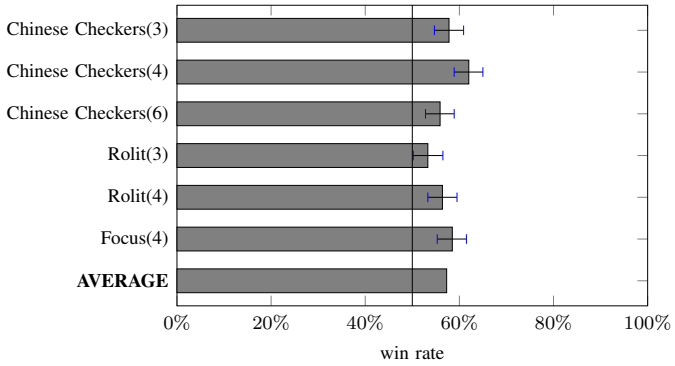


Fig. 4: Performance of MCTS with opponent move abstraction vs. vanilla MCTS. 250ms per move.

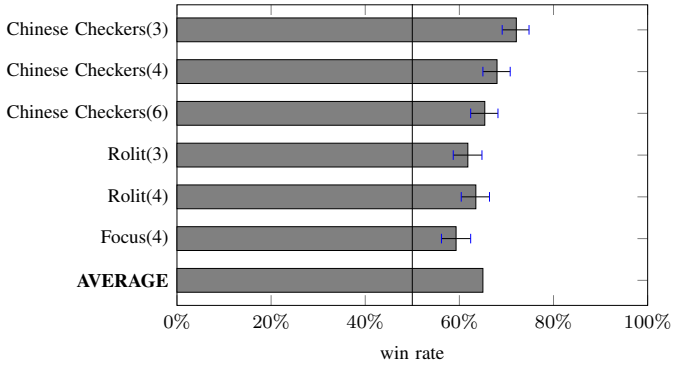


Fig. 5: Performance of MCTS with both progressive widening and opponent move abstraction vs. MCTS with only progressive widening. 250ms per move.

Our results agree with the main result for the comparison of MCTS and Paranoid/BRS at 250ms per move in [7], despite implementation differences and different enhancements used in their versions of MCTS and BRS.

Fig. 7 demonstrates the performance of MCTS-OMA against the baselines. Even though there is a modest improvement of the average win rates, now 53.1% against Paranoid and 37.0% against BRS+, six out of twelve comparisons still show MCTS-OMA to be the weaker player. Opponent move abstraction does not find enough generalizable information for the root player in the tree, even though it can still generalize for some opponent players (for example generalizing over the root player's first move at tree depth 1 when selecting a move at tree depth 2) and thus generate a small boost.

In Fig. 8, the results of MCTS-PW against Paranoid and BRS+ are shown. The average win rates are here raised to 71.9% against Paranoid and 57.0% against BRS+, which means MCTS-PW is much better than MCTS-OMA; the increased focus on moves heuristically assumed to be good results in much deeper and more effective search trees. However, three out of twelve comparisons still cannot show MCTS-PW to be significantly stronger than BRS+ (3- and 4-player Chinese Checkers as well as 4-player Rolit).

Fig. 9 finally supports the main point of this paper again:

That progressive widening and opponent move abstraction are ideally used together. While the win rates against Paranoid and BRS+ improved only by 4.8% on average when OMA was added to vanilla MCTS, adding OMA to MCTS-PW increased these win rates by 10.8%. MCTS-OMA-PW wins 79.6% of games against Paranoid search and 70.9% against BRS+, and is significantly stronger than both of the minimax-based baselines in all test domains at the 99.999% confidence level.

VI. Discussion and Future Research

In this paper, we introduced an approach tackling the problem of weak MCTS performance in multiplayer games, in particular at short time settings. Based on an analysis of the strengths of competing, minimax-based algorithms Paranoid and BRS+, we developed a combination of the existing soft pruning technique *progressive widening* and the newly proposed online generalization technique *opponent move abstraction*. We empirically demonstrated the synergy of the two techniques, and showed how they significantly outperform both baselines in all six test domains when used together in MCTS-OMA-PW, at the previously challenging time setting of 250ms per move.

The main idea of our approach is focusing your search as much as possible on yourself, and only as much as necessary on the opponents. This is achieved by growing a tree selectively and deeply enough to find your own next follow-up moves (PW), and then generalizing their values across the tree (OMA).

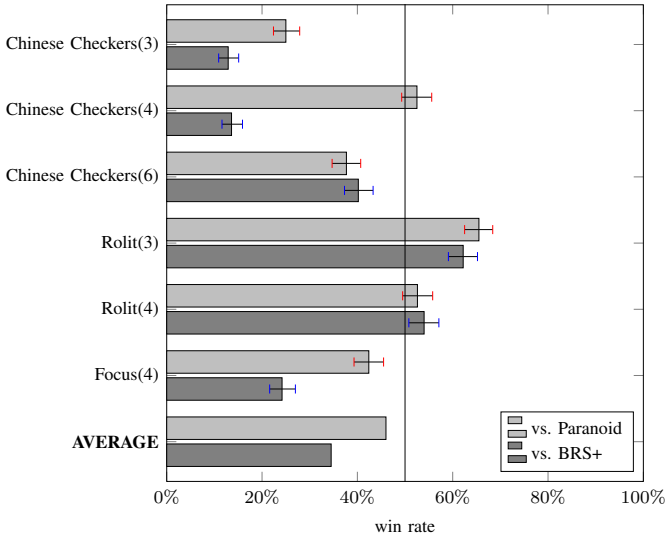


Fig. 6: Performance of vanilla MCTS vs. Paranoid and BRS+. 250ms per move.

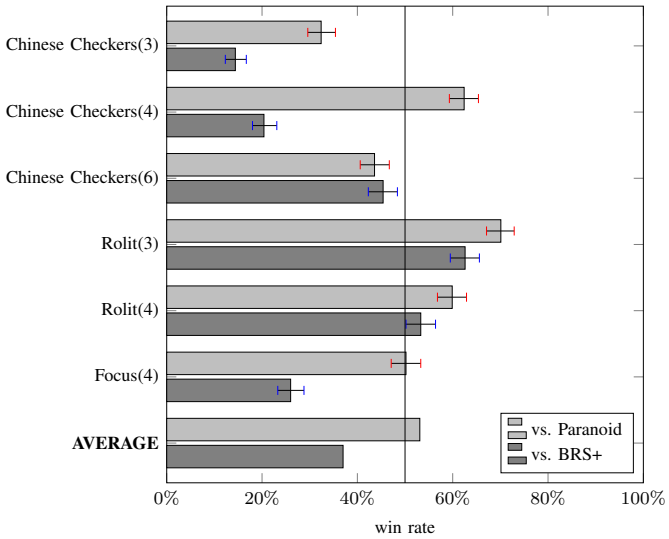


Fig. 7: Performance of MCTS with opponent move abstraction vs. Paranoid and BRS+. 250ms per move.

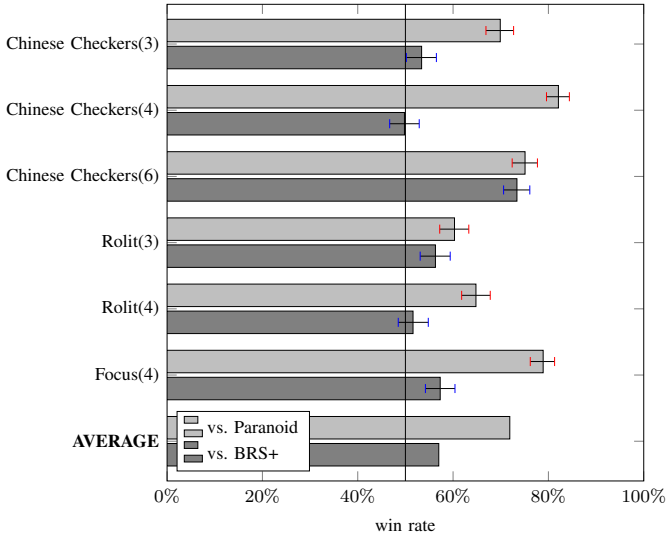


Fig. 8: Performance of MCTS with progressive widening vs. Paranoid and BRS+. 250ms per move.

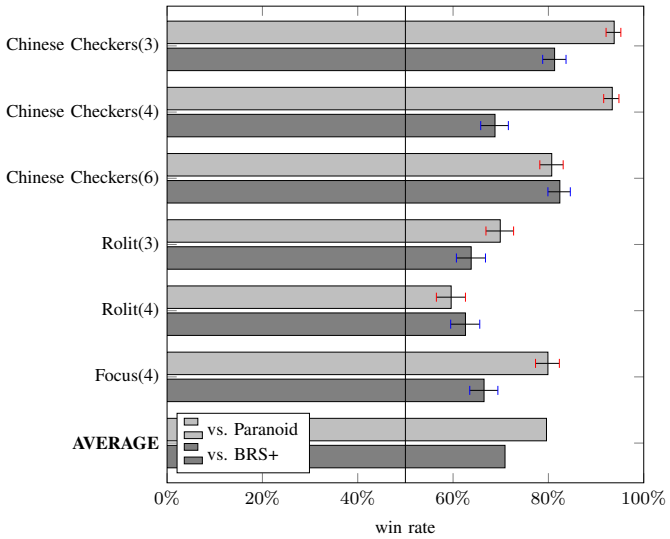


Fig. 9: Performance of MCTS with progressive widening and opponent move abstraction vs. Paranoid and BRS+. 250ms per move.

As a result, MCTS-OMA-PW is able to discover multi-move plans, without getting lost in the exponential explosion of mostly irrelevant opponent moves in between.

We see several perspectives for future work. First, PW or OMA are not necessarily restricted to deterministic, zero-sum, perfect-information, turn-taking, or board games, despite these being convenient testbeds. In fact, board games with their highly condensed environments and strong player interactions might be the worst case for this kind of approach; it could work even better in domains that are in some sense more similar to the real world, so that most actions have much more localized effects, and interactions are less common. We therefore see promising applications in for example collaborative, simultaneous-action multi-robot coordination problems.

Second, we expect benefits from integrating the approach presented here with an Expert Iteration [8] or Zero learning framework [4] – both because such frameworks could provide MCTS-OMA-PW with state-of-the-art, automatically learned state evaluators and move sorters instead of the hand-coded ones used in this paper, but also because MCTS-OMA-PW could potentially provide such frameworks with a powerful policy improvement operator at very short time settings.

Third, in this paper we compared one specific form of online generalization for MCTS to unenhanced minimax-type algorithms. Of course online generalization can also be used by minimax, e.g. in the history heuristic [24] or killer moves [25]. Both minimax and MCTS can use several forms of online generalization based on multiple different learning contexts, also simultaneously, and it remains to be seen which one is the most effective. We used a static move ordering scheme for PW, for example, while a dynamic one might allow the knowledge acquired by OMA to be fed back into improving PW for additional gains. The most interesting and challenging goal here might be the automatic end-to-end learning of an optimal online generalization technique.

ACKNOWLEDGMENTS

This work is part of the project *Flexible Assets Bid Across Markets* (FABAM, project number TEUE117015), funded within the Dutch Topsector Energie / TKI Urban Energy by Rijksdienst voor Ondernemend Nederland (RvO).

REFERENCES

[1] H. Finnsson, "Generalized Monte-Carlo Tree Search Extensions for General Game Playing," in *Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012)*, J. Hoffmann and B. Selman, Eds. AAAI Press, 2012.

[2] D. P. Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couëtoux, J. Lee, C. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, 2016.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the Game of Go without Human Knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-Play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[5] N. R. Sturtevant and R. E. Korf, "On Pruning Techniques for Multi-Player Games," in *17th National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, H. A. Kautz and B. W. Porter, Eds. AAAI Press / The MIT Press, 2000, pp. 201–207.

[6] M. P. D. Schadd and M. H. M. Winands, "Best Reply Search for Multiplayer Games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 1, pp. 57–66, 2011.

[7] P. A. M. Nijssen, "Monte-Carlo Tree Search for Multi-Player Games," Ph.D. dissertation, Department of Knowledge Engineering, Maastricht University, 2013.

[8] T. Anthony, Z. Tian, and D. Barber, "Thinking Fast and Slow with Deep Learning and Tree Search," *CoRR*, vol. abs/1705.08439, 2017.

[9] R. Coulom, "Computing Elo Ratings of Move Patterns in the Game of Go," *ICGA Journal*, vol. 30, no. 4, pp. 198–208, 2007.

[10] G. M. J. B. Chaslot, M. H. M. Winands, J. v. d. Herik, J. W. H. M. Uiterwijk, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.

[11] H. Baier and M. Kaisers, "Opponent-Pruning Paranoid Search," in *2020 International Conference on the Foundations of Digital Games (FDG 2020)*, Accepted.

[12] M. Esser, M. Gras, M. H. M. Winands, M. P. D. Schadd, and M. Lanctot, "Improving best-reply search," in *8th International Conference on Computers and Games (CG 2013)*, ser. Lecture Notes in Computer Science, H. J. van den Herik, H. Iida, and A. Plaat, Eds., vol. 8427. Springer, 2013, pp. 125–137.

[13] M. Gras, "Multi-Player Search in the Game of Billabong," Master's thesis, Department of Knowledge Engineering, Maastricht University, 2012.

[14] P. A. M. Nijssen and M. H. M. Winands, "Search Policies in Multi-Player Games," *ICGA Journal*, vol. 36, no. 1, pp. 3–21, 2013.

[15] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. P. Liebana, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[16] L. Kocsis and C. Szepesvári, "Bandit Based Monte-Carlo Planning," in *17th European Conference on Machine Learning (ECML 2006)*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Springer, 2006, pp. 282–293.

[17] H. Finnsson and Y. Björnsson, "Learning Simulation Control in General Game-Playing Agents," in *Twenty-Fourth AAAI Conference on Artificial Intelligence, (AAAI 2010)*, M. Fox and D. Poole, Eds. AAAI Press, 2010.

[18] M. J. W. Tak, M. H. M. Winands, and Y. Björnsson, "N-Grams and the Last-Good-Reply Policy Applied in General Game Playing," *IEEE Transastions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 73–83, 2012.

[19] S. Gelly and D. Silver, "Combining Online and Offline Knowledge in UCT," in *Twenty-Fourth International Conference on Machine Learning (ICML 2007)*, ser. ACM International Conference Proceeding Series, Z. Ghahramani, Ed., vol. 227. ACM, 2007, pp. 273–280.

[20] C. Luckhart and K. B. Irani, "An Algorithmic Solution of N-Person Games," in *5th National Conference on Artificial Intelligence*, T. Kehler, Ed. Morgan Kaufmann, 1986, pp. 158–162.

[21] F. Teytaud and O. Teytaud, "Creating an Upper-Confidence-Tree Program for Havannah," in *12th International Conference on Advances in Computer Games (ACG 2009)*, ser. Lecture Notes in Computer Science, H. J. van den Herik and P. Spronck, Eds., vol. 6048. Springer, 2009, pp. 65–74.

[22] S. Gelly and D. Silver, "Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

[23] N. R. Sturtevant, "An Analysis of UCT in Multi-Player Games," *ICGA Journal*, vol. 31, no. 4, pp. 195–208, 2008.

[24] J. Schaeffer, "The History Heuristic," *ICGA Journal*, vol. 6, no. 3, pp. 16–19, 1983.

[25] S. G. Akl and M. M. Newborn, "The Principal Continuation and the Killer Heuristic," in *1977 ACM Annual Conference*, J. S. Ketchel, H. Z. Kriloff, H. B. Burner, P. E. Crockett, R. G. Herriot, G. B. Houston, and C. S. Kitto, Eds. ACM, 1977, pp. 466–473.