Improving Best-Reply Search

Markus Esser, Michael Gras, Mark H.M. Winands,
Maarten P.D. Schadd and Marc Lanctot

Games and AI Group, Department of Knowledge Engineering,
Maastricht University, The Netherlands
markus.esser1@rwth-aachen.de,michael.gras@gmx.net,m.schadd@gmail.com,
{m.winands,marc.lanctot}@maastrichtuniversity.nl

Abstract. Best-Reply Search (BRS) is a new search technique for game-tree search in multi-player games. In BRS, the exponentially many possibilities that can be considered by opponent players is flattened so that only a single move, the best one among all opponents, is chosen. BRS has been shown to outperform the classic search techniques in several domains. However, BRS may consider invalid game states. In this paper, we improve the BRS search technique such that it preserves the proper turn order during the search and does not lead to invalid states. The new technique, BRS$^+$, uses the move ordering to select moves at opponent nodes that are not searched. Empirically, we show that BRS$^+$ significantly improves the performance of BRS in Four-Player Chess, leading to winning 8.3% to 11.1% more games against the classic techniques max^n and Paranoid, respectively. When BRS$^+$ plays against max^n, Paranoid, and BRS at once, it wins the most games as well.

1 Introduction

Research in the field of artificial intelligence has enjoyed immense success in the area of two-player zero-sum games of perfect information. Famous examples of such progress include IBM's Deep Blue vs. Kasparov [1], self-play learning to reach master level in Backgammon [11], and solving the game of Checkers [8].

Interest in abstract, deterministic multi-player games (> 2 players) has grown [5, 7, 10], but the amount of research in this area remains relatively small in comparison to that in the two-player setting. This is partly due to the fact that there are no worst-case equilibrium guarantees, but also to the added complexity introduced by more than two players.

In this paper, we propose a multi-player search technique, called BRS$^+$, which improves on a previous search technique called Best-Reply Search (BRS) [7]. In BRS, the root (search) player enumerates each of his moves, but search at opponents' nodes is restricted: only one of the opponents is allowed to act, the others must pass sequentially and the best single decision among all opponents is chosen to be played against the root player. As a result, the turn sequence is flattened so that the decisions strictly alternate between the root player and an opponent player. Collapsing the opponents' decisions in this way can lead

to invalid game states due to inconsistent turn order. For example, passing is not allowed in many classic games studied by AI researchers. Also, this modified search allows the root player to make more moves along the search path as all of the opponents are combined. Despite these problems, the computational load of the search is reduced and BRS has been shown to work well in practice in several different multi-player games [3, 6, 7].

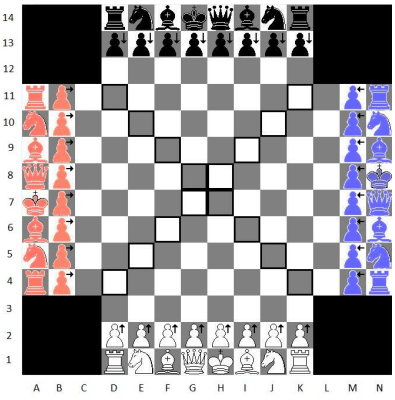
The main contribution of this paper is a reformulation of BRS that ensures only valid states are reached and that proper turn sequence is preserved, while still retaining the benefit of reduced computational complexity. This variant, called BRS$^+$, selects moves using move orderings rather than passing. We show that BRS$^+$ performs significantly better than BRS and the classic multi-player search techniques max^n [5] and Paranoid [10] in Four-Player Chess [4, 12].

The organization of the paper is as follows. First, we introduce the game of Four-Player Chess in Section 2. Then we formalize our problem and describe previous work in Section 3. We introduce BRS$^+$ and analyze its complexity in Section 4. We show results from a variety of experiments in Section 5. Finally, we conclude the paper and discuss potential future work in Section 6.

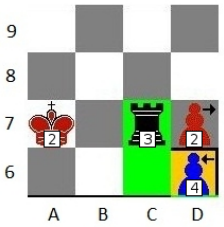
2 Four-Player Chess

Four-Player Chess is an extension of the classic game to four players. Its initial position is shown in Figure 1a. The rules we use here are adapted from the Paderborn rule set [4, 12]. Players strictly alternate turns clockwise, starting with White (1) to move, then Red (2), then Black (3), then Blue (4). Most rules from the two-player game remain unchanged, and the main differences are:

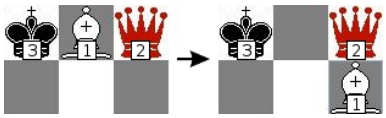
– Pawns are allowed to *bend-off* into a new direction at the diagonals such that the distance to promotion is kept the same as normal. This allows promotions on all sides. Once a pawn has bent-off, its direction can no longer change.



(a) Initial Board



(b) En-Passant Double Capture



(c) King Hanging

Fig. 1: Four-Player Chess

- En-passant can sometimes be combined with a capture move resulting in two captured pieces. For example, Player 2 moves a pawn that enables an en-passant move for Player 4 and Player 3 moves a piece to the en-passant capture square. In Figure 1b, Player 2 moves the pawn two spaces, Player 3 moves the rook, enabling an *en-passant double capture* for Player 4.
- Players can be eliminated in two ways: either they are check-mated at the beginning of the turn, or their king is hanged. The king hanging occurs for a player when that player cannot respond to his king being put in check due to turn order. In Figure 1c, if Player 1 moves his bishop and Player 2 is next to play, Player 3 is eliminated immediately. Once a player is eliminated, all of his pieces are removed from the board.
- The winner is the last player standing.

The game is made to be finite by forcing the usual draw-by-repetition and 50-move rule, each being unchanged from the standard 2-player version.

3 Multi-player Search

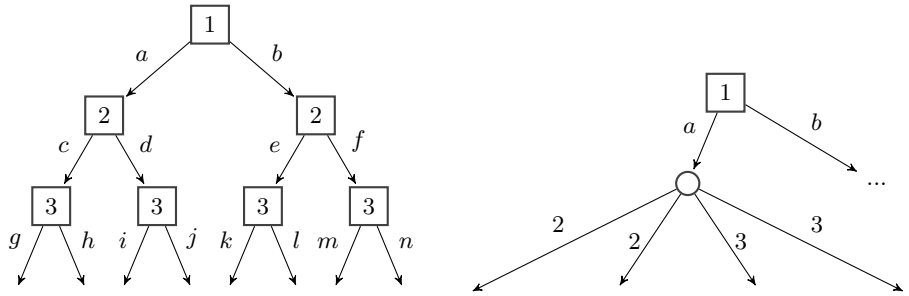
This section discusses search techniques for deterministic multi-player games. First, the classic search techniques max^n and Paranoid are described in Subsection 3.1. Next, Best-Reply Search (BRS) is introduced in Subsection 3.2.

3.1 Max^n and Paranoid Search

A finite game of perfect information can be described by a tuple $(\mathcal{N}, \mathcal{S}, \mathcal{Z}, \mathcal{A}, \mathcal{T}, P, u_i, h_i, s_0)$. The player set $\mathcal{N} = \{1, \ldots, n\}$ contains player labels and by convention a player is denoted $i \in \mathcal{N}$. The state space $\mathcal{S}$ is a finite, non-empty set of states, with $\mathcal{Z} \subseteq \mathcal{S}$ denoting the finite, non-empty set of terminal states. The move set $\mathcal{A}$ is a finite and non-empty. The utility functions $u_i : \mathcal{Z} \mapsto [v_{\min}, v_{\max}] \subseteq \mathbb{R}$ gives the utility of Player i, with v_{min} and $v_{\max}$ denoting the minimum and maximum possible utility, respectively. The heuristic evaluation functions $h_i : \mathcal{S} \mapsto [v_{\min}, v_{\max}]$ return a heuristic value of a state. In this paper, we assume constant-sum games: $\forall z \in \mathcal{Z}, \sum_{i \in \mathcal{N}} u_i(z) = k$, for some constant k. The player index function $P : \mathcal{S} \to \mathcal{N}$ returns the player to act in a given non-terminal state s, or a null value when its argument is a terminal state. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ describes the successor state given a state s and a move chosen from the available moves $\mathcal{A}(s) \subseteq \mathcal{A}$. We will also refer to the null or pass move as $\emptyset$. The game starts in an initial state s_0 and with player $P(s_0)$ to act. Finally, the tuple $u(s) = (u_1(s), \cdots, u_n(s))$ and $h(s)$ are defined in a similar way.

There are two classic techniques used for game tree search in multi-player games: max^n [5], and Paranoid [10]. Given a root state s, max^n searches to a fixed depth, selecting the move that maximizes the acting player's individual utility at each internal node $s \in \mathcal{S} \setminus \mathcal{Z}$, defined for move a at state s by:

$$V_d(s,a) = \begin{cases} u(s') & \text{if } s' \in \mathcal{Z} \\ h(s') & \text{if } s' \notin \mathcal{Z}, d = 0 \\ \max^i_{a' \in \mathcal{A}(s')} V_{d-1}(\mathcal{T}(s', a'), a') & \text{otherwise,} \end{cases} \quad (1)$$



(a) Original Game Tree (b) Transformed BRS Game Tree

Fig. 2: An example of how BRS transforms the multi-player search tree. In the BRS tree, the edges labeled 2 assume a pass move by Player 3 and vice versa.

where $i = P(s)$, $\max^i$ returns a tuple that maximizes the i^{th} value, $s' = \mathcal{T}(s, a)$ is the successor state when choosing move a in state s, and d is the depth to reach from s. When the depth is set to the height of the game tree, $\max^n$ does not use heuristic values and, with a sufficient tie-breaking rules, the final move choice belongs to an equilibrium strategy profile. In practice, this is often impossible due to the computational time requirement and hence usually small values for d are chosen.

Unlike $\max^n$, in Paranoid all the opponents collude against the root player. Paranoid can be described similarly using Equation 1 by changing i to always refer to the root player, and adding $\min^i_{a' \in \mathcal{A}(s')} V_{d-1}(s', a')$ if $s' \notin \mathcal{Z}, d > 0, P(s) \neq i$. In practice, consecutive opponent nodes can be thought of as one big opponent node, where each meta-move corresponds to a sequence of opponent moves. Hence the game effectively becomes a two-player game, and $\alpha\beta$-style pruning can be applied.

Assuming a uniform branching factor b and depth d, the worst-case running time required for both classic search techniques is $O(b^d)$ since all nodes may require visiting. In the best case, Paranoid takes time $O(b^{d(n-1)/n})$ in an n-player game [10]. In practice, Paranoid tends to outperform $\max^n$ since it searches deeper due to taking advantage of more pruning opportunities.

There are variants and hybrids of these techniques. For example, ProbMax^n incorporates beliefs from opponent models into the choice of decision made by $\max^n$. The Comixer technique models potential coalitions that can be formed from the current state of the game and chooses a move based on a mixed recommendation from each coalition [4]. MP-Mix decides on which search technique to invoke depending on how far the the leading player is perceived to be from the second placed player [13].

3.2 Best-Reply Search

In Best-Reply Search (BRS), instead of enumerating every move at each of the opponents' nodes independently in sequence, the opponents' nodes and moves

are merged into one opponent decision node [7]. This is similar to Paranoid with one key difference: the moves at the opponent nodes belong to *one* of the $(n-1)$ opponents. The particular move chosen at the opponent decision node is one that minimizes the root player's utility. As a result, the nodes visited strictly alternate between the root player's (max) nodes and opponent decision (min) nodes. In other words, only a single opponent chooses a move and the other opponents pass. The tree transformation process is depicted in Figure 2. Referring to Figure 2a, in general $\{g,h\} \neq \{i,j\}$ and similarly $\{k,l\} \neq \{m,n\}$.

Suppose now Player 1 chooses a and Player 2 ignores his legal moves and passes (plays $\emptyset$), there are no longer two distinct nodes that belong to Player 3. The game has then reached a state $s_{a,\emptyset} = \mathcal{T}(\mathcal{T}(s,a),\emptyset)$. As result, there is a single move set $\mathcal{A}(s_{a,\emptyset})$ available to Player 3. Define $s_{a,c}$ and $s_{a,d}$ similarly. To simplify the analysis, we assume uniform game trees, so in this example $|\mathcal{A}(s_{a,\emptyset})| = |\mathcal{A}(s_{a,c})| = |\mathcal{A}(s_{a,d})| = b$. Clearly, there will be duplicate moves at the opponent nodes in the BRS tree in Figure 2b, which need not be searched more than once. In fact, the opponent who is allowed to search will have b moves. There are $(n-1)$ opponents, therefore there will be $b(n-1)$ unique opponent choices to consider at opponent nodes rather than b^{n-1} in the case of Paranoid and max^n. As a result, BRS requires less time for a depth d search than Paranoid and max^n. However, this benefit comes at the cost of approximation error since collapsing the opponents' decisions in this way can lead to invalid game states due to inconsistent turn order, as passing is not allowed in many abstract games. Also, this modified search allows the root player to make more moves as all of the opponents are combined. Despite these problems, the computational load of the search is reduced and BRS has been shown to work well in practice in several different multi-player games such as Billabong [3], Blokus [6], Chinese Checkers [6, 7], and Focus [6, 7].

4 Generalized Best-Reply Search

In this section we present a reformulation of Best-Reply Search (BRS) by suggesting a different transformation of the game tree. The resulting BRS$^+$ tree is an augmented sub-graph of the original game tree. Then, applying the usual Paranoid-style search over in this new tree results in BRS$^+$ (Subsection 4.1). Next, a complexity analysis is given in Subsection 4.2.

4.1 Generalized BRS and BRS$^+$

To generalize BRS, the tree is transformed in such a way that a regular search results in one opponent enumerating all of his available moves and the other opponents being constrained to play a special move. How the special move is chosen depends on the specific rule used, which we elaborate on below. Define $\phi(s)$ to be a *special move*: a mapping to a pass move or a *regular move* from available moves $\mathcal{A}(s)$, but it is labeled differently since it is distinguished from the other moves. To simplify notation, we drop s and refer to mapped move as ϕ.

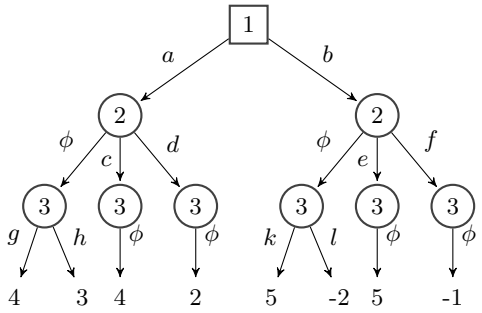


Fig. 3: An example of how the generalized BRS models the multi-player search tree, with payoffs belonging to the root player.

Given a multi-player game tree denote i the root player. States $s \in \mathcal{S}$ such that $P(s) = i$ remain unchanged. At states s such that $P(s) \neq i$, denote hist(i, s) the sequence of moves taken from s_i to s, where s_i denotes the most recent[1] state such that $P(s_i) = i$. If hist(i, s) contains exactly two regular moves, then all the regular moves allowed have been taken along hist(i, s), so set $\mathcal{A}'(s) = \{\phi\}$. Otherwise, set $\mathcal{A}'(s) = \mathcal{A}(s) \cup \{\phi\}$ unless $\mathcal{T}(s, a) = i$ in which case the node remains unchanged[2]. This last condition is required to ensure that the number of regular moves along hist$(i, \mathcal{T}(s, a))$ is always equal to 2. This transformation with $i = 1$ applied to the game tree in Figure 2a is shown in Figure 3.

This construction with the special moves generalizes BRS. For example, if $\phi = \emptyset$ the generalized BRS tree is equivalent to the one in Figure 2b. We focus on mapping special moves to move ordering moves (MOMs). In this case, illegal states cannot be reached since ϕ maps to a move in $\mathcal{A}(s)$, and we refer to the algorithm as BRS$^+$. The trees are not actually transformed. The move sets are manipulated during the recursive search, as presented in Algorithm 1. The critical modification in the tree search is to ensure that for every sequence of moves starting and ending at the root player, exactly one regular move and $(n - 2)$ special moves are taken by the opponents. This is achieved by counting the regular moves between successive turns of the root player i using a parameter m in Algorithm 1.

Two different approaches can be taken for selecting a special move by move ordering. One can order the moves in a way that attacks the root player i, or order moves in a way that looks most promising to $P(s)$. The former *paranoid move ordering* can be too pessimistic, preferring to capture the root player's pawn instead of another opponent's queen. The latter *max^n move ordering* can prefer to capture another player's queen over the root player's bishop even though the root player is in a much better position to win the game. We analyze the effect of these specific move-ordering strategies in Section 5.

[1] By "most recent" we mean the path with the shortest such sequence of moves.
[2] We assume, without loss of generality, that the game has a strictly alternating turn order, so $P(\mathcal{T}(s, a))$ will be the same $\forall a \in \mathcal{A}(s)$.

```
1  GBRS(node s, depth d, count m, root player i)
2    if s ∈ 𝒵 then return u_i(s)
3    else if d = 0 then return h_i(s)
4    else
5        𝒜'(s) ← 𝒜(s)
6        Let U ← ∅ be the set of child values, and j be the player following P(s)
7        if P(s) = i then m ← 0
8        else if P(s) ≠ i and m = 2 then 𝒜'(s) ← {φ}
9        else if P(s) ≠ i and j ≠ i then 𝒜'(s) ← 𝒜(s) ∪ {φ}
10       for a ∈ 𝒜'(s) do
11           s' ← 𝒯(s, a); m' ← m if a = φ else m + 1
12           u' ← GBRS(s', d − 1, m', i)
13           U ← U ∪ {u'}
14       return max(U) if P(s) = i else min(U)
```

Algorithm 1: Generalized Best-Reply Search

Similarly to Paranoid, only the root player's payoff is being considered, so standard $\alpha\beta$ pruning can be applied in BRS and BRS$^+$. From Figure 3, after the left subtree is traversed the value of taking move a is assessed to be 2. When the opponents are minimizing in the right subtree and the -2 is found, the remaining branches can be pruned since the root player will never choose move b since move a already has a higher value. Finally, when the special moves are mapped to MOMs, transpositions will occur, which can be subsequently be pruned [3].

4.2 Complexity Analysis

To show the number of nodes expanded in BRS$^+$, we use a similar argument to the one for analyzing Paranoid [10]. Recall that, as is common in this setting, uniform trees are assumed and hence the branching factor, b, is a constant.

We start with the worst-case analysis. To analyze the complexity, we return to the original tree transformation depicted in Figure 2b (specifically, *not* the tree depicted in Figure 3.) The same argument used in Subsection 3.2 to merge the state and move sets applies when opponent moves are mapped using ϕ. We first analyze the depth reached in this transformed tree and then relate it to the true depth reached. Suppose depth D is reached in this tree, then $D/2$ levels of max nodes are expanded and $D/2$ levels of min nodes are expanded. For simplicity, we assume D is even. How does D relate to the true depth d? At a max node in the tree, one ply represents a decrease in true depth of 1. At a min node, one ply represents a decrease in true depth of $(n-1)$. Therefore $d = D/2 + (n-1)D/2 \Rightarrow D = 2d/n$. At max nodes, the branching factor is b. At min nodes, the branching factor is $b(n-1)$. Therefore, the number of nodes expanded is $f_{worst}(b,n,D) = b \cdot b(n-1) \cdot b \cdot b(n-1) \cdot \cdots b(n-1)$, with $\frac{D}{2}$ occurrences of b and $\frac{D}{2}$ occurrences of $b(n-1)$. Formally, $f_{worst}(b,n,D) =$

$$b^{\frac{D}{2}} \cdot (b(n-1))^{\frac{D}{2}} = b^{\frac{D}{2}} \cdot b^{\frac{D}{2}} \cdot (n-1)^{\frac{D}{2}} = b^D \cdot (n-1)^{\frac{D}{2}} = b^{\frac{2d}{n}} \cdot (n-1)^{\frac{d}{n}},$$

so the time taken for a depth d search is $O(b^{\frac{2d}{n}}(n-1)^{\frac{d}{n}})$. Therefore, since $n > 2$, BRS$^+$ expands asymptotically fewer nodes as $d \to \infty$ than max^n and Paranoid in the worst case when $b > n - 1$, which is true in most large games.

The best-case analysis follows the original BRS best-case analysis [7]. There are two extremes. Suppose the recursive search finds the lowest payoff value (a loss) at every leaf visited. Every min node searches a single move in this case because the rest of the moves can be pruned due to a loss being the worst possible payoff. Max nodes still require enumerating each move in the hopes of finding something better than a loss. Using the same logic as above, this requires $b^{\frac{D}{2}} = b^{\frac{d}{n}}$ expansions. The other extreme is that every leaf visited is a win. In this case the roles are reversed and every move at max nodes after the first can be pruned but all the moves at min nodes need to be enumerated. This requires $(b(n-1))^{\frac{D}{2}} = b^{\frac{d}{n}} \cdot (n-1)^{\frac{d}{n}}$. The search technique must specify a strategy for all players, so if max has a win then min must have a loss, and vice versa. Therefore, the best case is the worst of these extremes, and hence $O(b^{\frac{d}{n}}(n-1)^{\frac{d}{n}})$ node expansions are required. As a result, compared to the worst case complexity, the exponent of b is halved.

While these results are encouraging, we make two key observations. First, the overhead involved with computing the move ordering move may not be negligible. Secondly, the performance critically depends on the quality of the move ordering. If the move suggested is worse than passing, BRS$^+$ could perform worse than its predecessor, BRS. We investigate these points further in the following section.

5 Experimental Results

When directly comparing two multi-player search techniques, there are $2^4 - 2 = 14$ possible seating arrangements (four of the same type of player is not allowed). When comparing four techniques there are $4! = 24$ possible seating arrangements and 36 when exactly three techniques are present. To ensure fairness, all of the experiment results are performed in batches where each batch consists of every possible seating arrangement for the number of players being compared. Therefore, the number of games per experiment is always a multiple of the number of seating arrangements for the specified number of players.

The Four-Player Chess engine is written in Java, and the experiments were run on machines equipped with AMD Opteron 2.4 GHz processors. The evaluation function and search enhancements are described in [2]. All the times listed in this section are in milliseconds, and intervals represent 95% confidence intervals.

5.1 Baseline Experiments

As a first experiment we ran 1440 games of the base search techniques max^n, Paranoid, and BRS. The results are listed in Table 1. As we can see from the results, Paranoid and BRS outperform max^n considerably, with Paranoid performing slightly better than BRS.

These results were unexpected, so we ran another experiment to compare each individual search technique directly. BRS beat $\max^n$ in 56.0% ± 4.2% of the games, BRS won against Paranoid in 58.9% ± 4.1% of the games, and Paranoid beat $\max^n$ in 59.0% ± 2.9% of the games [2]. These results suggest that while BRS is able to beat Paranoid, Paranoid does a slightly better job of exploiting $\max^n$ than BRS does, so when all three are present Paranoid has a slight edge over BRS.

5.2 Move Ordering Experiments in BRS$^+$

In the next series of experiments we compare the two different approaches for selecting the special moves in BRS$^+$: paranoid vs. $\max^n$ move ordering. Recall the differences from Section 4. The paranoid move ordering prefers immediate captures of the root player's pieces while $\max^n$ move ordering considers captures of all opponents' pieces. The results of the two compared directly, as mentioned above, are shown in Table 2. As we see, the Paranoid move ordering performs better. This is somewhat expected as BRS$^+$ more closely resembles Paranoid than $\max^n$. We will test the performance of each type of static move ordering against the other search techniques in Subsection 5.3.

We also optimize our static move ordering with *attacker tie-breaking*. Suppose a player is able to capture an opponent's piece using a pawn or a queen. Often it is better to take the piece using a pawn, in case the capturing piece is susceptible to a counter-capture. To assess the benefit of this optimization, we ran 1120 games of BRS$^+$ with and without it. The optimized version won 52.6% ± 3.0% of the games. While the benefit is small, it still seems to have a positive effect on the move ordering, so it is included from here on. Finally, if two different moves capturing the same piece have the same attacker piece value, then values stored by the history heuristic (see below) are used as the final tie-breaker.

If there are no capture moves available from the static move ordering, then a dynamic move ordering may be applied. As is standard in search implementations, an iterative deepening version of BRS$^+$ is used with incrementally increasing depth d. Whenever a node is searched, the best move is stored at the node's entry in the transposition table. The first dynamic move ordering is then to consult the transposition table for this previously found best move. In addition, we try the killer-move heuristic and history heuristic. Unfortunately, none of

Time	Games	BRS	$\max^n$	Paranoid
1000 ms	1440	38.4% ± 2.6%	19.6% ± 2.1%	42.1% ± 2.6%

Table 1: Performance results of BRS vs. $\max^n$ vs. Paranoid experiments.

Time	Games	BRS$^+$ ($\max^n$ MO)	BRS$^+$ (paranoid MO)
1000 ms	1120	42.3% ± 2.9%	57.7% ± 2.9%

Table 2: Direct comparison using different static move orderings for BRS$^+$.

Static move ordering variant	max^n	Paranoid	BRS
BRS^+ (max^n MO)	59.3% (± 4.1%)	56.3% (± 4.2%)	53.0% (± 4.2%)
BRS^+ (Paranoid MO)	64.3% (± 4.0%)	70.0% (± 3.8%)	54.2% (± 3.0%)
BRS	56.0% (± 4.2%)	58.9% (± 4.1%)	-

Table 3: Comparison of BRS variants vs. max^n, Paranoid.

the dynamic move orderings seemed to improve performance significantly when enabled independently. In 560 games with a time limit of one second, enabling transposition tables leads to a win rate of 49.6% ± 4.2%. Similarly enabling killer moves and the history heuristic leads to a win rate of 50.8% ± 4.2% and 51.7% ± 4.2%, respectively. We believe this is due to the instability of decision-making in games with more than two players, but more work is required for a concrete analysis on this point.

In the rest of the experiments, only the static move ordering is used with the attacker tie-breaking optimization.

5.3 Performance of BRS^+ vs. BRS, max^n, Paranoid

The first experiment compares the performance of BRS^+ with either paranoid or max^n move ordering when playing against one of the classic algorithms. Using a time limit of 1000 ms, each variant of BRS^+ is player against each previous algorithm separately [2]. The results are shown in Table 3. The first thing to notice is that BRS^+ is winning in every instance. Clearly, the paranoid static move ordering is the better choice for BRS^+ as it is superior to the max^n ordering in all of our experiments. Finally, the performance of BRS^+ using the paranoid move ordering versus the classic algorithms increases significantly compared to the BRS from baseline experiments in Section 5.1, increasing by 11.1% (from to 58.9% to 70%) against Paranoid and 8.3% (from 56% to 64.3%) against max^n. This last results shows a clear benefit of BRS^+ over BRS when making direct comparisons to max^n and Paranoid.

Naturally, we also want know how BRS^+ performs when played in the multi-player setting against the other two search techniques. First, we ran a similar experiment to the baseline experiments in the three algorithm setting including BRS^+, Paranoid, and max^n. The results are presented in Table 4. Again, the performance of BRS^+ is improved significantly compared to BRS, from 38.4% to 49.4% compared to results from Table 1. In addition, BRS^+ becomes the decisive winner by a margin of 16.3%.

Finally, we ran an experiment including all four algorithms. The results are shown in Table 5. Again, BRS^+ is the winner, beating the second place player

Time	Games	BRS^+	max^n	Paranoid
1000 ms	720	49.4% ± 3.7%	17.6% ± 2.8%	33.1% ± 3.5%

Table 4: BRS^+ vs. max^n vs. Paranoid.

Time	Games	BRS$^+$	BRS	max^n	Paranoid
1000 ms	960	32.9% ± 3.0%	26.5% ± 2.8%	17.7% ± 2.5%	22.9% ± 2.7%
5000 ms	960	35.4% ± 3.1%	22.7% ± 2.7%	17.4% ± 2.4%	24.5% ± 2.8%

Table 5: BRS$^+$ vs. BRS vs. max^n vs. Paranoid.

Time	Positions	BRS$^+$	max^n	Paranoid
1000 ms	200	6.105	3.075	4.115
5000 ms	200	7.125	3.895	4.845

Table 6: Average depths reached by search algorithms.

Time	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 10$
1000 ms	51.6 ± 2.1%	53.3 ± 2.1%	51.2 ± 2.1%	52.2 ± 2.1%	52.3 ± 2.1%
5000 ms	51.0 ± 3.7%	54.1 ± 3.8%	56.2 ± 3.8%	57.6 ± 3.7%	55.8 ± 4.1%

Table 7: Performance of BRS$^+$ with Rand-Top-k optimization.

roughly by a 5% gap. This gap more than doubles when the time limit is increased to five seconds. To confirm our expectation that BRS$^+$ is searching deeper, we ran an additional experiment to compute the depth reached by each technique in the time allowed on a suite of 200 board positions. The results are presented in Table 6. These results show that, on average, BRS$^+$ reaches 1.99 to 2.28 ply deeper than Paranoid for these time settings.

5.4 Rand-Top-k Move Ordering Experiment

Because the performance depends heavily on the move ordering, we also tried randomly choosing among the top k moves in the move ordering. This adds some variation in the moves considered by the opponents, which may lead to more robustness against bias in the move ordering. We compare the performance of Rand-Top-k directly by playing games with it enabled and disabled. The results are shown in Table 7. It seems that randomizing over the top k moves further improves the performance of BRS$^+$. These initial investigations suggest that the effect may not be smooth in the value of k at the lower time setting.

6 Conclusions and Future Research

In this paper, we introduced a new multi-player search technique, BRS$^+$, which can avoid invalid states and preserve the proper turn order during its search. BRS$^+$ expands asymptotically fewer nodes than the classic algorithms max^n and Paranoid as $d \to \infty$ leading to deeper searches. When BRS$^+$ uses move ordering moves at opponent nodes that are not searched, its performance critically depends on the move ordering used. Through experiments in Four-Player Chess, we show that a paranoid static move ordering outperforms a static max^n

move ordering in BRS$^+$. Finally, in all experiments the performance of BRS$^+$ is significantly higher than its predecessor, BRS.

For future work, we aim to compare our algorithms to the Comixer and MP-Mix algorithms mentioned in Section 3. In addition, we hope to extend the algorithm to the multi-player Monte-Carlo Tree Search [6, 9] setting and apply it to other multi-player games such as Chinese Checkers, Blokus, and Focus as well as Hearts, which was not formerly playable by BRS.

Acknowledgments. We would like to thank Nathan Sturtevant for the suggestion of Rand-Top-k and Pim Nijssen for his help with multi-player search algorithms. This work is partially funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938.

References

1. Campbell, M., Hoane Jr., A.J., Hsu, F.: Deep Blue. Artificial Intelligence 134(1-2), 57–83 (2002)
2. Esser, M.: Best-Reply Search in Multi-Player Chess. Master's thesis, Department of Knowledge Engineering, Maastricht University (2012)
3. Gras, M.: Multi-Player Search in the Game of Billabong. Master's thesis, Department of Knowledge Engineering, Maastricht University (2012)
4. Lorenz, U., Tscheuschner, T.: Player modeling, search algorithms and strategies in multi-player games. In: van den Herik, H.J., Hsu, S., Hsu, T., Donkers, H.H.L.M. (eds.) Proceedings of Advances in Computer Games. Lecture Notes in Computer Science (LNCS), vol. 4250, pp. 210–224. Springer-Verlag, Berlin, Heidelberg (2006)
5. Luckhart, C.A., Irani, K.B.: An algorithmic solution of n-person games. In: AAAI'86. pp. 158–162 (1986)
6. Nijssen, J.A.M., Winands, M.H.M.: Search policies in multi-player games. ICGA Journal 36(1), 3–21 (2013)
7. Schadd, M.P.D., Winands, M.H.M.: Best reply search for multi-player games. IEEE Transactions on Computational Intelligence and AI in Games 3(1), 57–66 (2011)
8. Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S.: Checkers is solved. Science 317(5844), 1518–1522 (2007)
9. Sturtevant, N.R.: An analysis of UCT in multi-player games. ICGA Journal 31(4), 195–208 (2008)
10. Sturtevant, N.R., Korf, R.E.: On pruning techniques for multi-player games. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000). pp. 201–207 (2000)
11. Tesauro, G.: Temporal difference learning and td-gammon. Communications of the ACM 38(3), 58–68 (1995)
12. Tscheuschner, T.: Four-person chess (Paderborn rules) (2005), `http://www2.cs.uni-paderborn.de/cs/chessy/pdf/4Prules.pdf`
13. Zuckerman, I., Felner, A.: The MP-Mix algorithm: Dynamic search strategy selection in multiplayer adversarial search. IEEE Transactions on Computational Intelligence and AI in Games 3(4), 316–331 (2011)