SEARCH POLICIES IN MULTI-PLAYER GAMES [1]

J. A. M. Nijssen and Mark H. M. Winands [2]

Maastricht, The Netherlands

ABSTRACT

In this article we investigate how three multi-player search policies, namely max^n, paranoid, and Best-Reply Search, can be embedded in the MCTS framework. The performance of these search policies is tested in four different deterministic multi-player games with perfect information by running self-play experiments. We show that MCTS with the max^n search policy overall performs best.

Furthermore, we introduce a multi-player variant of the MCTS-Solver. We propose three update rules for solving nodes in a multi-player MCTS tree. The experimental results show that the multi-player variant of the MCTS-Solver is a genuine improvement for MCTS in multi-player games.

1. INTRODUCTION

Monte-Carlo Tree Search (MCTS) is a best-first search technique that can easily be extended from two-player to multi-player games (Sturtevant, 2008). Different search policies can be applied that indicate how the children are selected and how the results are backpropagated in the tree. The basic multi-player MCTS algorithm applies a search policy which is analogous to the max^n search tree. In a max^n tree, each player maximizes his own score (Luckhardt and Irani, 1986). In the standard MCTS algorithm, a similar principle is applied. Each player tries to maximize his own win rate, while not considering the win rates of the opponents. Similar to the classic minimax framework, it is possible to apply the paranoid and BRS search policies to MCTS as well.

This article investigates the max^n (Luckhardt and Irani, 1986), paranoid (Sturtevant and Korf, 2000) and Best-Reply Search (BRS) (Schadd and Winands, 2011) search policies in the MCTS framework. They are called MCTS-max^n, MCTS-paranoid and MCTS-BRS, respectively. Their performance is tested in four different multi-player games, namely Chinese Checkers, Focus, Rolit and Blokus. Furthermore, these MCTS variants are compared to the minimax-based search techniques.

Next, MCTS-max^n is modified so that it is able to prove positions and therefore play tactical lines better. The Monte-Carlo Tree Search Solver (MCTS-Solver) (Winands, Björnsson, and Saito, 2008) concept is applied in MCTS-max^n. Experiments are performed in the sudden-death game Focus.

The article is structured as follows. First, in Section 2 related work on search techniques in multi-player games is discussed. Section 3 explains how the paranoid and BRS search policies can be applied in the MCTS framework. A brief overview of the test domains is given in Section 4. Next, Section 5 provides the experimental results of the different search policies. A background of the MCTS-Solver is given in Section 6. MCTS-Solver for multi-player games is introduced in Section 7. Subsequently, in Section 8 we provide the experimental results for the multi-player MCTS-Solver. Finally, the conclusions and an overview of possible future research directions are given in Section 9.

[1]This article is a revised and updated version of the following two contributions: (1) Nijssen, J.A.M. and Winands, M.H.M. (2012). An Overview of Search Techniques in Multi-Player Games. *Computer Games Workshop at ECAI 2012*, pp. 50–61, Montpellier, France; (2) Nijssen, J.A.M. and Winands, M.H.M. (2011). Enhancements for Multi-Player Monte-Carlo Tree Search. *Computers and Games (CG 2010)* (eds. H.J. van den Herik, H. Iida, and A. Plaat), Vol. 6515 of *LNCS*, pp. 238–249, Springer-Verlag, Berlin, Germany.

[2]Games and AI Group, Department of Knowledge Engineering, Faculty of Humanities and Sciences, Maastricht University, E-mail:{pim.nijssen,m.winands}@maastrichtuniversity.nl

2. SEARCH TECHNIQUES IN MULTI-PLAYER GAMES

In this section, we provide a background on search techniques for multi-player games. The minimax-based algorithms max^n, paranoid and BRS are discussed in Subsection 2.1. The application of MCTS to multi-player games is described in Subsection 2.2.

2.1 Minimax-Based Search Algorithms

The traditional algorithm for searching in trees for multi-player games is *max^n* (Luckhardt and Irani, 1986). This technique is an extension of minimax search to multi-player games. In the leaf nodes of the search tree, each player is awarded a value, based on a heuristic evaluation function. These values are stored in a tuple of size N, where N is the number of players. The sum of the values for all players may be constant. When backing up the values in the tree, each player always chooses the move that maximizes his score. A disadvantage of max^n is that $\alpha\beta$ pruning is not possible. Luckhardt and Irani (1986) proposed shallow pruning, which is an easy and safe way to achieve some cutoffs. It is only possible if the sum of the values for all players is bound to a fixed maximum. With shallow pruning, the asymptotic branching factor is $\frac{1+\sqrt{4b-3}}{2}$. However, in the average case the asymptotic branching factor is b, which means that $O(b^d)$ nodes are investigated (Korf, 1991). An example of a max^n tree with shallow pruning is displayed in Figure 1. In this example, $N = 3$ and the sum of the values for all players is 10. Other pruning techniques include last-branch pruning and speculative pruning (Sturtevant, 2003b). Another disadvantage of max^n lies in the fact that the assumption is made that opponents do not form coalitions to reduce the player's score. This can lead to too optimistic play. The optimism can be diminished by making the heuristic evaluation function more paranoid or by using the paranoid tie breaker rule (Sturtevant, 2003a).

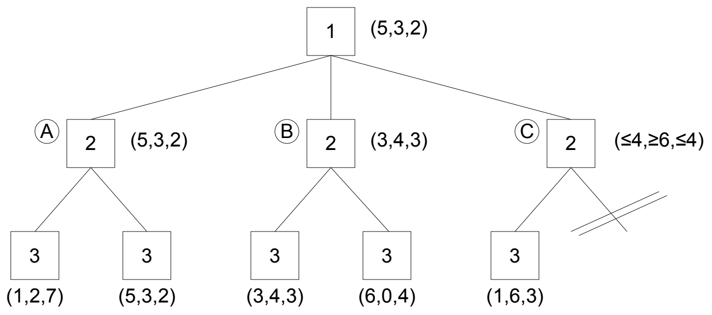


Figure 1: A max^n search tree with shallow pruning.

Paranoid search was first mentioned by Von Neumann and Morgenstern (1944) and was later investigated by Sturtevant and Korf (2000). It assumes that all opponents have formed a coalition against the root player. Using this assumption, the game can be reduced to a two-player game where the root player is represented in the tree by MAX nodes and the opponents by MIN nodes. The advantage of this assumption is that $\alpha\beta$-like deep pruning is possible in the search tree, allowing deeper searches in the same amount of time. An example of a paranoid search tree is provided in Figure 2. In the best case, $O(b^{\frac{N-1}{N}d})$ nodes are investigated in a paranoid search tree. This is a generalization of the best case for two-player games. Because more pruning is possible than in max^n, paranoid may outperform max^n because of the larger lookahead (Sturtevant, 2003a). The disadvantage of paranoid search is that, because of the often incorrect paranoid assumption, the player may become too defensive. Furthermore, if the complete game tree is evaluated, a paranoid player may find that all moves are losing, because winning is often not possible if all opponents have formed a coalition. In general, the deeper a paranoid player searches, the more pessimistic he becomes (Saito and Winands, 2010).

Recently, Schadd and Winands (2011) proposed a new algorithm for playing multi-player games, namely *Best-Reply Search* (BRS). This technique is similar to paranoid search, but instead of allowing all opponents to make a move, only one opponent is allowed to do so. This is the player with the best counter move against the root player. Similar to paranoid, $\alpha\beta$ pruning is possible in BRS. In the best case, BRS investigates $O\left((b(N-1))^{\lceil \frac{2d}{N} \rceil / 2}\right)$ nodes. The advantage of this technique is that more layers of MAX nodes are investigated, which leads to more long-term planning. Furthermore, this algorithm is less pessimistic than the paranoid algorithm, and less opti-

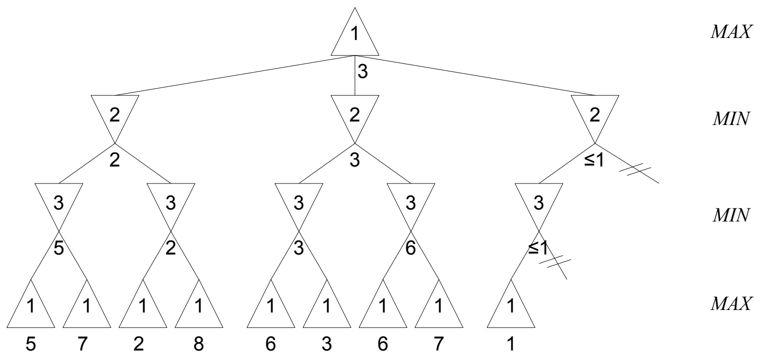


Figure 2: A paranoid search tree with $\alpha\beta$ pruning.

mistic than the max^n algorithm. Compared to paranoid, only one opponent performs a counter move against the root player, instead of all opponents, and compared to max^n, BRS does not use the optimistic assumption that the opponents are only concerned with their own value. The disadvantage is that, if passing is not allowed, invalid positions, i.e., illegal or unreachable positions, are taken into account. This is the reason why in some games, such as trick-based card games like Bridge or Hearts, BRS cannot be applied. To overcome this disadvantage, Esser (2012) and Gras (2012) proposed a modification of BRS, where the other opponents, rather than skip their turn, play a move based on domain knowledge. Esser (2012) showed that, using this enhancement, the modified version of BRS was able to outperform standard BRS in four-person chess. In this article, we only investigate the performance of the standard BRS algorithm. An example of a BRS tree is provided in Figure 3. Note that the moves of all opponents are compressed into one layer. The numbers next to the edges indicate to which players the corresponding moves belong.

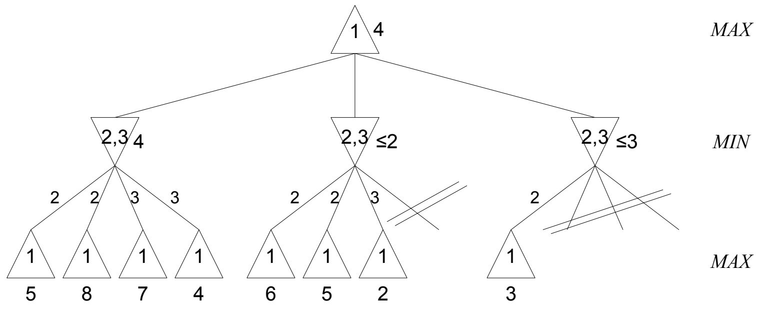


Figure 3: A BRS search tree with $\alpha\beta$ pruning.

2.2 Monte-Carlo Tree Search for Multi-Player Games

Monte-Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006; Coulom, 2007) is a best-first search technique that gradually builds up a search tree, guided by Monte-Carlo simulations.The MCTS algorithm consists of four phases (Chaslot *et al.*, 2008): selection, expansion, playout, and backpropagation. By repeating these four phases iteratively, the search tree is constructed gradually. In the *selection* phase, the search tree is traversed, starting from the root, using the *UCT selection strategy* (Kocsis and Szepesvári, 2006). In the *expansion* phase, one node is added to the tree (Coulom, 2007). During the *playout* phase, moves are played, starting from the position represented by the newly added node, until the game is finished. In the *backpropagation* phase, the result of the playout is propagated back along the previously traversed path up to the root node.

Applying MCTS to multi-player games is quite straightforward (Sturtevant, 2008). The difference with the application to two-player games is that, after each playout is finished, instead of a single value, a tuple of N values, is backpropagated in the tree. Each value in the tuple corresponds to the score achieved by one player. Each value is in $[0, 1]$, where 1 corresponds to a win and 0 to a loss. In the case of a draw between several players, 1 point is divided evenly among the winners. The selection strategy remains the same. Each player tries to maximize his own win rate using the standard UCT formula. The child i with the highest value v_i is selected as follows (Formula 1).

$$v_i = \bar{x_i} + C\sqrt{\frac{\ln(n_p)}{n_i}} \tag{1}$$

In this formula, v_i is the UCT value of child i, $\bar{x_i}$ is the win rate of the current player, n_p is the number of visits of the current node, and n_i is the number of visits of child i. C is the exploration constant.

The search policy in this implementation is similar to max^n, where each player tries to maximize his own score. In the next section we show that it is also possible to apply different search policies, such as paranoid or BRS, in the multi-player MCTS framework.

MCTS was first applied to deterministic perfect information multi-player games by Sturtevant (2008) and Cazenave (2008). Sturtevant (2008) provided an analysis of the MCTS algorithm in Chinese Checkers and perfect information variants of Spades and Hearts. He showed that, in Chinese Checkers, MCTS wins over 90% of the games against max^n and paranoid, and that paranoid significantly outperforms max^n. In these experiments, all players were allowed 250,000 node expansions per move. He found that with fewer expansions there is not enough data for UCT to form accurate estimates. For instance, MCTS only wins 16.7% of games against paranoid when both algorithms are only allowed 1600 node expansions per move. In other domains, MCTS plays on a par with existing programs in the game of Spades, and better than existing programs in Hearts. Furthermore, Sturtevant (2008) proved that UCT is able to compute a mixed equilibrium in a multi-player game tree. Cazenave (2008) applied MCTS to multi-player Go. He introduced a technique called Paranoid UCT. In his design of the Paranoid UCT, the paranoia is modelled in the playouts, while the MCTS tree is traversed in the usual way. He tested the performance of Paranoid UCT against a RAVE player in three-player Go, and found that Paranoid UCT performs better.

3. ALTERNATIVE SEARCH POLICIES IN MULTI-PLAYER MCTS

In this section, we propose the application of the paranoid and BRS search policies in MCTS. First, MCTS-paranoid is explained in more detail in Subsection 3.1. MCTS-BRS is discussed in Subsection 3.2.

3.1 MCTS-Paranoid

The idea of using a paranoid search policy in MCTS was suggested by Cazenave (2008), however he did not implement or test it. When applying the paranoid search policy in MCTS, the structure of the tree remains intact, however all nodes in the opponents' layers are changed into MIN nodes. All nodes in the root player's layers remain MAX nodes and the standard UCT formula is applied. When using paranoid search in MCTS, the opponents use a different UCT formula. Instead of maximizing their own win rate, they try to minimize the win rate of the root player. In the MIN nodes of the tree, the following modified version of the UCT formula is used. The child i with the highest value v_i is selected as follows (Formula 2).

$$v_i = (1 - \bar{x_i}) + C\sqrt{\frac{\ln(n_p)}{n_i}} \tag{2}$$

Similar to the UCT formula (Formula 1), $\bar{x_i}$ denotes the win rate of node i. n_i and n_p denote the total number of times child i and its parent p have been visited, respectively. C is a constant, which balances exploration and exploitation.

The major difference with the standard UCT formula is that, at the MIN nodes, $\bar{x_i}$ does not represent the win rate at child i of the current player, but of the root player. Essentially, $(1 - \bar{x_i})$ indicates the win rate of the coalition of the opponents. Analogous to paranoid in the minimax framework, the opponents do not consider their own win rate.

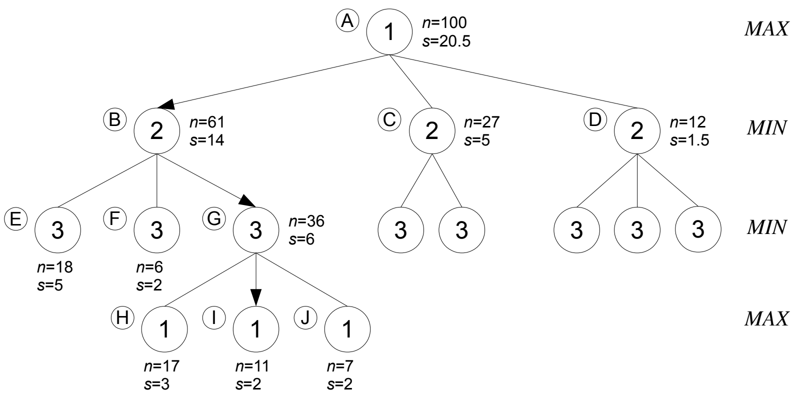


Figure 4: Example of an MCTS-paranoid tree.

An example of a paranoid tree in the MCTS framework for a three-player game is provided in Figure 4. In this example, 100 playouts have been simulated so far. For each node, n indicates how often the node has been visited so far and s indicates the cumulative score of the root player. The value $\bar{x_i}$ of node i is computed using $\bar{x_i} = \frac{s_i}{n_i}$. The root player applies the standard UCT formula (Formula 1) to select a child. Assuming $C = 0.2$, the UCT values for nodes B, C and D are:

$$\begin{aligned}
v_B &= \frac{14}{61} + 0.2\sqrt{\frac{\ln 100}{61}} &&\approx 0.284\\
v_C &= \frac{5}{27} + 0.2\sqrt{\frac{\ln 100}{27}} &&\approx 0.268\\
v_D &= \frac{1.5}{12} + 0.2\sqrt{\frac{\ln 100}{12}} &&\approx 0.249
\end{aligned}$$

In this iteration, the root player chooses node B. From this node, the first opponent can choose between nodes E, F and G. Because the paranoid search policy is applied, the modified UCT formula is applied to select the next child. The UCT values of the children of node B are calculated as follows.

$$\begin{aligned}
v_E &= (1 - \frac{5}{18}) + 0.2\sqrt{\frac{\ln 61}{18}} &&\approx 0.818\\
v_F &= (1 - \frac{2}{6}) + 0.2\sqrt{\frac{\ln 61}{6}} &&\approx 0.832\\
v_G &= (1 - \frac{6}{36}) + 0.2\sqrt{\frac{\ln 61}{36}} &&\approx 0.901
\end{aligned}$$

Even though the first opponent may have a higher win rate at node E or F, he chooses node G to try to minimize the win rate of the root player. After selecting node G, the UCT values for the second opponent are calculated in a similar way.

$$\begin{aligned}
v_H &= (1 - \frac{3}{17}) + 0.2\sqrt{\frac{\ln 36}{17}} &&\approx 0.915\\
v_I &= (1 - \frac{2}{11}) + 0.2\sqrt{\frac{\ln 36}{11}} &&\approx 0.932\\
v_J &= (1 - \frac{2}{7}) + 0.2\sqrt{\frac{\ln 36}{7}} &&\approx 0.857
\end{aligned}$$

For the second opponent, node I is chosen. If node I is fully expanded, then the next child is chosen using the standard UCT formula. Once a leaf node is reached, the playout is performed in the standard way. When the playout is finished, the value s of nodes A, B, G and I is increased with the score of the root player and n is, similar to the standard MCTS algorithm, increased by 1.

3.2 MCTS-BRS

With the BRS policy in MCTS, all opponents' layers are compressed into one layer, similar to BRS in the minimax framework. The standard UCT formula is applied in the root player's layers and the paranoid UCT formula is used in the opponents' layers.

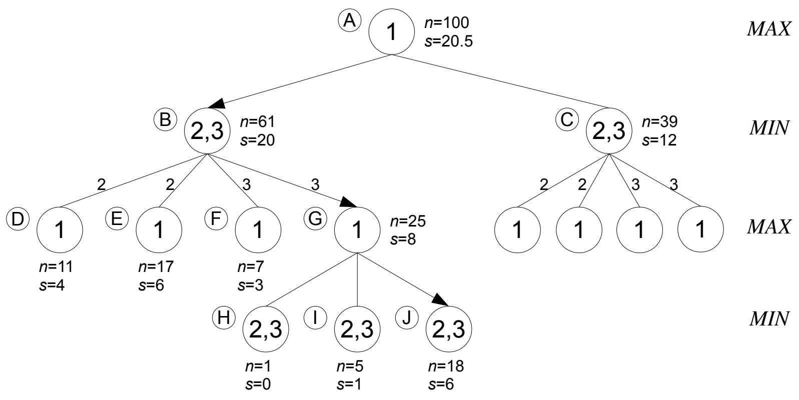


Figure 5: Example of an MCTS-BRS tree.

An example of an MCTS-BRS tree is given in Figure 5. Similar to MCTS-paranoid and MCTS-max^n, the root player selects a move using the standard UCT formula as follows.

$$v_B = \frac{20}{61} + 0.2\sqrt{\frac{\ln 100}{61}} \approx 0.383$$
$$v_C = \frac{12}{39} + 0.2\sqrt{\frac{\ln 100}{39}} \approx 0.376$$

The root player selects node B. Next, the moves of the two opponents are compressed into one layer. Nodes D and E represent positions that are reached after moves by Player 2, i.e., the first opponent, and nodes E and F represent positions that are reached after moves by Player 3, i.e., the second opponent. The paranoid UCT formula is applied to select a move for one of the opponents as follows.

$$v_D = (1 - \frac{4}{11}) + 0.2\sqrt{\frac{\ln 61}{11}} \approx 0.759$$
$$v_E = (1 - \frac{6}{17}) + 0.2\sqrt{\frac{\ln 61}{17}} \approx 0.745$$
$$v_F = (1 - \frac{3}{7}) + 0.2\sqrt{\frac{\ln 61}{7}} \approx 0.725$$
$$v_G = (1 - \frac{8}{25}) + 0.2\sqrt{\frac{\ln 61}{25}} \approx 0.761$$

Node G is selected, so the first opponent skips a move and only the second opponent plays a move at this point. Next, the root player selects a move again using the standard UCT formula.

$$\begin{aligned}
v_H &= \frac{0}{1} + 0.2\sqrt{\frac{\ln 25}{1}} &&\approx 0.358\\
v_I &= \frac{1}{5} + 0.2\sqrt{\frac{\ln 25}{5}} &&\approx 0.360\\
v_J &= \frac{6}{18} + 0.2\sqrt{\frac{\ln 25}{18}} &&\approx 0.418
\end{aligned}$$

After selecting node J, this procedure continues until a leaf node is reached. Once a leaf node is reached, the tree is expanded and the playout is performed in the standard way. The result of the playout is backpropagated in a similar way as in MCTS-paranoid.

4. TEST DOMAINS

The experiments in this article are performed in four different deterministic perfect information multi-player games: Chinese Checkers, Focus, Rolit, and Blokus.

Chinese Checkers is a board game that can be played by two to six players. The game was invented in 1893 and has since then been released by various publishers under different names. Chinese Checkers is a popular test domain for multi-player search techniques. For instance, Sturtevant (2008) and Nijssen and Winands (2011) used the game for investigating MCTS in multi-player games. Schadd and Winands (2011) used Chinese Checkers to test the performance of BRS against paranoid and max^n. It is also frequently used as one of the test domains in General Game Playing (Clune, 2007; Finnsson, 2012; Tak, Winands, and Björnsson, 2012). The initial board position for six-player Chinese Checkers is provided in Figure 6(a).

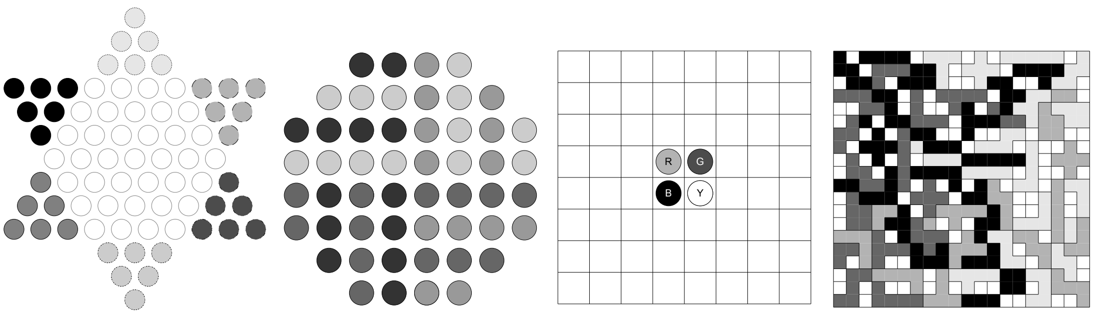
Focus is a multi-player strategy board game, which was described by Sid Sackson in 1969. This game has also been released under the name *Domination*. Focus has been used as a test bed for investigating the performance of Progressive History and the Multi-Player MCTS-Solver by Nijssen and Winands (2011) and for comparing the performance of BRS to paranoid and max^n by Schadd and Winands (2011). Figure 6(b) shows the initial position for four-player Focus.

Rolit, published in 1997 by Goliath, is a multi-player variant of the two-player game Othello. This game was introduced in 1975. It is similar to a game invented around 1880, called Reversi. This game was invented by either Lewis Waterman or John W. Mollett. At the end of the 19th century it gained much popularity in England, and in 1898 games publisher Ravensburger started producing the game as one of its first titles. Saito and Winands (2010) applied Paranoid Proof-Number Search to solve the two-, three-, and four-player variants of Rolit on 4×4 and 6×6 boards, and the four-player variant on the 8×8 board. They used this technique to calculate the minimum score each player is guaranteed, assuming optimal play and that all opponents have formed a coalition. It was also used as one of the domains for testing the performance of BRS, paranoid and max^n by Schadd and Winands (2011). The initial position for Rolit is given in Figure 6(c).

Blokus is a four-player tile placement game developed by Bernard Tavitian in 2000. Because Blokus is a relatively new game, little research has been performed in this domain. Shibahara and Kotani (2008) used the two-player variant of Blokus, Blokus Duo, as a test domain for combining final score and win rate in Monte-Carlo evaluations. A finished game of Blokus is displayed in Figure 6(d).

5. EXPERIMENTAL RESULTS FOR SEARCH POLICIES

In this section, we describe the experiments for the search policies and their results. First, Subsection 5.1 gives an overview of the experimental setup. Subsection 5.2 provides a comparison between the three minimax-based search techniques for multi-player games: max^n, paranoid and BRS. Next, Subsection 5.3 investigates how the paranoid and BRS search policies in MCTS compare against the max^n policy in the MCTS framework. A comparison between the strongest minimax-based technique and the different MCTS-based techniques is given in Subsection 5.4. In the final set of experiments, the strongest MCTS-based technique are compared to the three minimax-based techniques in Subsection 5.5.



(a) Chinese Checkers (b) Focus (c) Rolit (d) Blokus

Figure 6: The four games investigated in this article.

5.1 Experimental Setup

For the MCTS-based players, the UCT exploration constant C is set to 0.2. The selection phase is enhanced with Progressive History (Nijssen and Winands, 2011), which is a domain-independent technique for biasing the selection strategy when the number of playouts is low. In this article, we set the influence factor W to 5. Furthermore, ϵ-greedy playouts (Sutton and Barto, 1998; Sturtevant, 2008) are applied. For each move in the playout, a random move is played with a probability of ϵ (here 0.05). Otherwise, static move ordering is applied to select a move. These values were achieved by systematic testing.

All minimax-based players use a transposition table (Greenblatt, Eastlake, and Crocker, 1967) with the DEEP replacement scheme (Breuker, Uiterwijk, and van den Herik, 1996), and static move ordering. Furthermore, the paranoid and BRS players use killer moves (Akl and Newborn, 1977) and the history heuristic (Schaeffer, 1983). Finally, for the max^n player, shallow pruning is applied (Sturtevant and Korf, 2000). To allow shallow pruning, the scores retrieved from the heuristic evaluation function are normalized, such that the sum of the scores for all players is 1.

A description of the applied domain knowledge for the static move ordering and the heuristic evaluation function is given in Nijssen and Winands (2012).

The program is written in Java. The experiments were run on a server consisting of AMD OpteronT 2.2 GHz processors. There are various ways to assign two or three types of players to the different seats in multi-player games (Sturtevant, 2003a). Table 1 shows in how many ways the player types can be assigned. Only the configurations where at least one instance of each player type is present are considered. There may be an advantage regarding the order of play and the number of instances of each player type. Therefore, each assignment is played multiple times until at least 1,000 games are played and each assignment was played equally often. All experiments are performed with 250 ms, 1000 ms, and 5000 ms thinking time per move, unless stated otherwise. The results are given with a 95% confidence interval.

Table 1: The number of ways two or three different player types can be assigned. The number between brackets is the number of games that are played per match.

Number of players	2 player types	3 player types
3	6 (1050)	6 (1050)
4	14 (1050)	36 (1044)
6	62 (1054)	540 (1080)

5.2 Comparison of Minimax-based Techniques

Before testing the max^n, paranoid and BRS search policies in MCTS, we first investigate how they perform when applied in the minimax framework. In the first set of experiments, we therefore match the three basic minimax-based players against each other. The win rates and the average search depths of the players in the different games are displayed in Table 2. In this series of experiments, we validate the results found by Schadd and Winands (2011), and extend the experiments with one more game (Blokus) and more variation in the number of players.

Table 2: Results of max^n vs. paranoid vs. BRS.

			Max^n		Paranoid		BRS	
Game	Players	Time (ms)	Win rate (%)	Depth (ply)	Win rate (%)	Depth (ply)	Win rate (%)	Depth (ply)
Chinese Checkers	3	250	1.1±0.6	3.04	24.8±2.6	4.44	74.1±2.6	4.75
		1000	1.0±0.6	3.41	20.5±2.4	5.11	78.6±2.5	5.44
		5000	1.4±0.7	4.15	21.6±2.5	5.75	76.9±2.5	6.72
Chinese Checkers	4	250	5.3±1.4	2.95	11.7±1.9	3.52	83.0±2.3	4.09
		1000	4.0±1.2	3.57	23.0±2.5	4.83	72.9±2.7	5.04
		5000	5.7±1.4	4.07	19.4±2.4	5.43	74.8±2.6	5.86
Chinese Checkers	6	250	15.0±2.1	2.88	13.9±2.1	3.34	71.1±2.7	3.53
		1000	12.2±2.0	3.85	13.1±2.0	4.10	74.1±2.6	4.74
		5000	16.9±2.2	4.13	12.8±2.0	4.59	69.5±2.7	5.12
Focus	3	250	4.4±1.2	3.58	35.7±2.9	4.27	59.9±3.0	4.34
		1000	3.8±1.2	4.06	28.6±2.7	4.88	67.6±2.8	5.03
		5000	3.8±1.2	4.63	27.3±2.7	5.26	69.0±2.8	5.93
Focus	4	250	9.4±1.8	3.34	17.5±2.3	3.55	73.1±2.7	4.15
		1000	7.0±1.5	3.81	24.0±2.6	4.66	69.0±2.8	4.86
		5000	6.7±1.5	4.39	27.9±2.7	5.23	65.4±2.9	5.36
Rolit	3	250	8.1±1.7	5.29	38.6±2.9	5.72	53.3±3.0	5.56
		1000	8.9±1.7	6.12	39.7±3.0	6.74	51.3±3.0	6.65
		5000	6.3±1.5	6.86	45.4±3.0	7.88	48.4±3.0	7.73
Rolit	4	250	15.9±2.2	4.81	41.5±3.0	5.48	42.5±3.0	5.01
		1000	14.7±2.1	5.48	42.7±3.0	6.38	42.6±3.0	5.90
		5000	14.9±2.2	6.39	42.2±3.0	7.28	42.9±3.0	7.08
Blokus	4	250	17.8±2.3	2.21	30.4±2.8	3.11	51.8±3.0	2.80
		1000	15.4±2.2	2.66	29.6±2.8	3.70	55.1±3.0	3.65
		5000	8.6±1.7	3.28	23.5±2.6	4.32	68.0±2.8	4.43

In Chinese Checkers, BRS is the best search technique by a considerable margin. In the variants with three, four and six players, BRS wins between 69.5% and 83.0% of the games with any time setting. In three-player and four-player Chinese Checkers, paranoid is significantly stronger than max^n. This is because paranoid can, on average, search more than 1 ply deeper and can therefore search a second layer of MAX nodes more often. In six-player Chinese Checkers, max^n is at least as strong as paranoid. In this variant, both max^n and paranoid usually do not reach a second layer of MAX nodes, as this requires a 7-ply search. BRS has a considerable advantage here, because this technique only requires a 3-ply search to reach a second layer of MAX nodes, which happens quite often. We note that for Chinese Checkers, max^n does not normalize the heuristic evaluation function and, as such, does not use shallow pruning. Empirical testing showed that this variant performs better in this game than the default approach, where the evaluation scores are normalized and shallow pruning is applied.

In Focus, again BRS is the best search technique and, similar to Chinese Checkers, reaches on average the highest search depth. Max^n performs quite poorly in Focus, where it never reaches a win rate of more than 10%.

In Rolit, the difference between BRS and paranoid is much smaller. In three-player Rolit, BRS is still significantly better than paranoid for short time settings, but with 5 seconds thinking time BRS and paranoid are equally strong. In the four-player variant, BRS and paranoid are on equal footing with any time setting. One of the possible reasons is that, contrary to Chinese Checkers and Focus, BRS does not reach a higher search depth than paranoid. This is true for all time settings.

Finally, in Blokus BRS achieves the highest win rate again. In this game the average search depth is lower than in the other games. This is because Blokus has, especially in midgame, a high branching factor which can go up to more than 500 legal moves. Furthermore, because the board is large, computing the legal moves for a player is quite time-consuming, which reduces the number of nodes that are investigated per second.

General remarks

The results show that among the three tested search techniques, max^n performs the least. In every game with any number of players and time setting, max^n has a significantly lower win rate that both paranoid and BRS.

The exception is six-player Chinese Checkers. Because there is only little pruning possible using the paranoid algorithm, max^n plays at least as strong as paranoid. We remark that if better pruning techniques are applied for max^n, this search technique may perform better in other game variants as well. Max^n also plays relatively well in Blokus, where all players have difficulty reaching a decent search depth. Only the BRS player can regularly reach a second level of MAX nodes. In most games, BRS is the best search technique. Overall, the BRS players can search slightly deeper than the paranoid players. The most notable exception is Rolit. In this game, the paranoid players can generally search slightly deeper and perform on a similar level as BRS. Overall, the experimental results are comparable with the results found by Sturtevant (2008) and Schadd and Winands (2011).

5.3 Comparison of MCTS-based Techniques

In the second set of experiments, the performance of the three different search policies in MCTS is tested. Each player uses a different policy: max^n (MCTS-max^n), paranoid (MCTS-paranoid) or BRS (MCTS-BRS). They are enhanced with ϵ-greedy playouts and Progressive History. The win rates and the median number of playouts per move are summarized in Table 3.

Table 3: Results of MCTS-max^n vs. MCTS-paranoid vs. MCTS-BRS.

			MCTS-max^n		MCTS-paranoid		MCTS-BRS	
Game	Players	Time (ms)	Win rate (%)	Playouts (median)	Win rate (%)	Playouts (median)	Win rate (%)	Playouts (median)
		250	40.2±3.0	1,007	28.5±2.7	1,003	31.3±2.8	994
Chinese Checkers	3	1000	51.7±3.0	4,318	19.9±2.4	4,368	28.4±2.7	4,257
		5000	62.1±2.9	22,693	10.8±1.9	22,765	27.0±2.7	22,163
		250	47.8±3.0	791	28.9±2.7	786	23.3±2.6	767
Chinese Checkers	4	1000	52.8±3.0	3,520	19.0±2.4	3,546	28.3±2.7	3,396
		5000	64.4±2.9	18,513	12.2±2.0	18,921	23.5±2.6	17,698
		250	46.9±3.0	623	28.2±2.7	635	24.8±2.6	595
Chinese Checkers	6	1000	54.4±3.0	2,792	20.7±2.4	3,033	24.9±2.6	2,725
		5000	61.2±2.9	14,948	14.1±2.1	18,787	24.7±2.6	14,151
		250	40.8±3.0	1,629	29.1±2.7	1,642	30.2±2.8	1,609
Focus	3	1000	42.9±3.0	6,474	26.1±2.7	6,668	31.0±2.8	6,382
		5000	48.7±3.0	32,987	18.8±2.4	34,446	31.7±2.8	31,990
		250	37.3±2.9	1,416	33.3±2.9	1,405	29.4±2.8	1,350
Focus	4	1000	41.2±3.0	6,310	26.1±2.7	6,619	32.8±2.8	5,945
		5000	52.3±3.0	33,618	18.8±2.4	37,693	28.9±2.7	31,299
		250	50.6±3.0	1,460	28.9±2.7	1,465	20.5±2.4	1,428
Rolit	3	1000	57.3±3.0	5,933	24.6±2.6	5,905	18.1±2.3	5,787
		5000	63.2±2.9	30,832	20.4±2.4	30,019	16.4±2.2	29,673
		250	43.6±3.0	1,496	31.4±2.8	1,497	25.0±2.6	1,409
Rolit	4	1000	50.0±3.0	6,064	27.5±2.7	6,034	22.5±2.5	5,651
		5000	56.5±3.0	31,689	20.8±2.5	30,977	22.7±2.5	28,818
		250	36.7±2.9	325	34.5±2.9	320	28.8±2.7	295
Blokus	4	1000	36.0±2.9	1,406	35.3±2.9	1,344	28.8±2.7	1,231
		5000	33.6±2.9	6,932	34.3±2.9	6,824	32.0±2.8	6,210

In Chinese Checkers, MCTS-max^n is with any number of players and with any time setting the strongest search technique. Overall, MCTS-BRS performs better than MCTS-paranoid. If more time is provided, MCTS-max^n performs relatively better than with lower time settings. With 250 ms thinking time, MCTS-max^n wins between 40.2% and 47.8% of the games, depending on the number of players. With 5 seconds of thinking time, the win rate increases to between 61.2% and 64.4%. MCTS-paranoid performs relatively worse with higher time settings, while MCTS-BRS remains stable. Furthermore, we note that there is overall no large difference between the median number of playouts per move between the different search policies. This is not only true in Chinese Checkers, but also in the three other games. Although, in Chinese Checkers, if the number of players increases, the median number of playouts drops.

In Focus, MCTS-max^n is the best technique as well, though its win rate generally is lower than in Chinese Checkers. With 250 ms thinking time, it performs only slightly better than MCTS-BRS and MCTS-paranoid in the three- and four-player variants. Similar to Chinese Checkers, however, MCTS-max^n performs relatively better with higher time settings. Its win rate increases to around 50% with 5 seconds thinking time, while especially MCTS-paranoid performs worse with this time setting.

In Rolit, MCTS-max^n is again the strongest of the three algorithms. In the three-player variant of Rolit, MCTS-max^n appears to play relatively better than in the four-player variant, while MCTS-BRS appears to play relatively better in four-player Rolit. Similar to Chinese Checkers and Focus, the performance of MCTS-max^n increases with more thinking time, while the performance of MCTS-paranoid decreases.

Finally, in Blokus there is no clear winner. With 250 ms and 1 second of thinking time, MCTS-max^n and MCTS-paranoid are equally strong, with MCTS-BRS slightly behind. With 5 seconds thinking time, the three players are all on the same footing and there is no significant difference between the players. Similar to the results in the previous set of experiments, in Blokus the smallest number of positions is explored. Again, this is caused by the time-consuming generation of moves.

Experiments with Vanilla MCTS

Because ϵ-greedy playouts and Progressive History alter the selection and the playout phase of the MCTS algorithm, we validate the previous experiments by rerunning them with Vanilla MCTS, i.e. with ϵ-greedy playouts and Progressive History disabled for all players. Only the experiments with 1 second of thinking time are repeated. The results are given in Table 4.

Table 4: Results of MCTS-max^n vs. MCTS-paranoid vs. MCTS-BRS without ϵ-greedy playouts and Progressive History.

		MCTS-max^n		MCTS-paranoid		MCTS-BRS	
Game	Players	Win rate (%)	Playouts (median)	Win rate (%)	Playouts (median)	Win rate (%)	Playouts (median)
Chinese Checkers	3	36.7±2.9	1,709	30.7±2.8	1,714	32.7±2.8	1,702
	4	48.7±3.0	2,412	23.7±2.6	2,396	27.7±2.7	2,369
	6	71.0±2.7	6,470	8.1±1.6	6,918	20.9±2.5	6,182
Focus	3	24.5±2.6	242	38.5±2.9	242	37.0±2.9	242
	4	29.0±2.7	427	35.5±2.9	426	35.4±2.9	422
Rolit	3	48.5±3.0	5,983	27.1±2.7	6,022	24.4±2.6	5,827
	4	49.1±3.0	6,443	26.5±2.7	6,473	24.3±2.6	5,970
Blokus	4	36.0±2.9	1,217	34.5±2.9	1,114	29.5±2.8	1,048

There are two striking results. First, the median number of playouts per move increases with the number of players in Chinese Checkers. This is in contrast with the results found in Table 3. This phenomenon is caused by the fact that the pieces move randomly on the board and that the game is finished when one of the home bases is filled. With more players, there are more home bases and more pieces on the board. As a result it takes, on average, less moves before one of the home bases is filled. Second, MCTS-max^n is outperformed by both MCTS-paranoid and MCTS-BRS in Focus. This may be caused by the low number of playouts per move. Because the moves in the playouts are chosen randomly, games can take a long time to finish. This result is in accordance with the results in Subsection 5.3, where we found that MCTS-paranoid performs relatively better and MCTS-max^n performs relatively worse if the number of playouts is lower. Also in Chinese Checkers, playouts take much longer than with ϵ-greedy playouts, except in the six-player variant.

In Rolit and Blokus, the average length of the playouts is similar to the previous set of experiments. This is because the length of these games is not dependent on the strategy of the players. A game of Rolit always takes 60 turns, and a game of Blokus never takes more than 84 turns. This may explain why the results in this set of experiments are comparable to those in Table 3.

General remarks

Overall, the results reveal that MCTS clearly performs best using the standard max^n search policy. Only in Blokus, MCTS-max^n is not significantly stronger than MCTS-paranoid and MCTS-BRS. Without ϵ-greedy playouts and Progressive History, MCTS-max^n is outperformed by MCTS-paranoid and MCTS-BRS in Focus.

This is different to the minimax framework, where paranoid and BRS significantly outperform max^n. There are two main reasons for this difference. First, paranoid and BRS perform well in the minimax framework because they increase the amount of pruning. Because $\alpha\beta$ pruning does not occur in MCTS, this advantage is non-existent in the MCTS framework. Second, in the minimax framework, BRS reduces the *horizon effect*. It allows more planning ahead because more layers of MAX nodes are investigated. In MCTS, the advantage of having more layers of MAX nodes in the search tree is considerably smaller. The horizon effect in MCTS is already diminished due to the playouts. An additional problem with MCTS-BRS is that, in the tree, invalid positions are investigated, which may reduce the reliability of the playouts.

The results also show that MCTS-max^n performs relatively better than the other two techniques if more time is provided. Especially MCTS-paranoid performs relatively worse with more thinking time. The reason for this may be that the paranoid assumption causes the player to become too paranoid with larger search depths, similar to paranoid in the minimax framework. In Blokus, the performance of the three different players is stable with different time settings. Finally, the results reveal that there is overall no large difference in the median number of playouts between the different players. This indicates that the different search policies do not produce a significantly different amount of overhead.

5.4 MCTS-Based Techniques versus BRS

The experiments in Section 5.3 revealed that MCTS-max^n is the best among the different MCTS variants. In the next series of experiments, this result is validated by comparing the three different search policies in MCTS against the best minimax-based search technique, BRS (cf. Subsection 5.2). The results are displayed in Table 5. The percentages indicate the win rate of each of the players against BRS.

Table 5: Win rates of the different MCTS-based techniques against BRS.

Game	Players	Time (ms)	MCTS-max^n win rate (%)	MCTS-paranoid win rate (%)	MCTS-BRS win rate (%)
		250	18.4±2.3	15.0±2.2	14.6±2.1
Chinese Checkers	3	1000	42.4±3.0	29.4±2.6	35.5±2.9
		5000	68.2±2.8	29.2±2.8	50.0±3.0
		250	24.5±2.6	16.7±2.3	18.1±2.3
Chinese Checkers	4	1000	57.7±3.0	45.5±3.0	48.0±3.0
		5000	77.6±2.5	47.1±3.0	65.8±2.9
		250	33.3±2.8	25.5±2.6	24.1±2.6
Chinese Checkers	6	1000	72.1±2.7	56.4±3.0	64.5±2.9
		5000	88.1±2.0	73.3±2.7	83.8±2.2
		250	37.1±2.9	32.2±2.8	34.2±2.9
Focus	3	1000	53.8±3.0	37.7±2.9	48.1±3.0
		5000	62.9±2.9	34.5±2.9	54.7±3.0
		250	42.3±3.0	37.0±2.9	39.6±3.0
Focus	4	1000	54.3±3.0	39.7±3.0	50.5±3.0
		5000	68.6±2.8	42.8±3.0	61.3±2.9
		250	74.1±2.6	65.3±2.9	58.6±3.0
Rolit	3	1000	84.6±2.2	69.8±2.8	68.0±2.8
		5000	87.0±2.0	68.7±2.8	69.0±2.8
		250	71.2±2.7	66.6±2.9	60.9±3.0
Rolit	4	1000	80.2±2.4	66.0±2.9	64.5±2.9
		5000	82.0±2.3	64.0±2.9	67.2±2.8
		250	57.8±3.0	56.2±3.0	57.5±3.0
Blokus	4	1000	77.4±2.5	80.9±2.4	79.9±2.4
		5000	90.5±1.8	89.1±1.9	88.0±2.0

In Chinese Checkers, the win rate of the MCTS players strongly depends on the thinking time. If 250 ms per move are provided, MCTS-max^n wins between 18.4% and 33.3% of the games against BRS, dependent on the number of players. With 5000 ms thinking time, the win rate lies between 68.2% and 88.1% against BRS. MCTS-paranoid and MCTS-BRS win significantly fewer games against BRS, which indicates that MCTS-max^n is a stronger player than MCTS-paranoid and MCTS-BRS. This is in accordance with the results found in Subsection 5.3.

In Focus, similar results are found. With a lower time setting, all MCTS-based opponents are significantly outperformed by BRS, while with 5000 ms of thinking time per move, the win rate increases to between 55% and 70% for MCTS-max^n and MCTS-BRS. MCTS-paranoid wins around or less than 40% of the games against BRS with most time settings.

In Rolit, the MCTS-based players perform well compared to BRS. In both the three- and four-player variant, MCTS-max^n wins more than 70% of the games with any time setting against BRS. Also MCTS-paranoid and MCTS-BRS win significantly more than 60% of the games against BRS. This again shows that Rolit is a difficult domain for BRS.

Finally, in Blokus, BRS is outperformed by the MCTS-based players as well. This is likely because BRS can only reach a limited search depth because of the high branching factor. The win rate of the three different MCTS players is similar, which again shows that the three different MCTS-based players are on equal footing in Blokus.

General remarks

The results in Table 5 show that MCTS-max^n is the strongest player against BRS. This result is in accordance with the results in Subsection 5.3. MCTS-paranoid and MCTS-BRS achieve a significantly lower win rate against BRS, except in Blokus. When comparing BRS to MCTS-max^n, for the low time settings BRS significantly outperforms MCTS-max^n in Focus and Chinese Checkers, while MCTS-max^n is stronger in Blokus and Rolit. With a higher time setting, MCTS-max^n becomes stronger than BRS in all games. This is not true for MCTS-paranoid, which performs worse than BRS in the three-player and four-player variants of Chinese Checkers and Focus, even with high time settings. Similar to the results found in Subsection 5.3, MCTS-paranoid does not benefit much from reaching larger search depths. MCTS-BRS does, however, benefit from higher time settings. It outperforms BRS in all game variants, except in three-player Chinese Checkers, where the two players are equally strong.

5.5 Minimax-based Techniques versus MCTS-max^n

In the next set of experiments we test the performance of the three minimax-based techniques against the strongest MCTS-based technique, MCTS-max^n. The win rates of max^n, paranoid and BRS against MCTS-max^n are given in Table 6. We note that the win rates in the column 'BRS' are the inverse of the win rates under 'MCTS-max^n' in Table 5, as these two columns both show the results of the matches between MCTS-max^n and BRS.

In Chinese Checkers, max^n and paranoid are much weaker than MCTS-max^n. This result was also found by Sturtevant (2008). BRS wins more games against MCTS-max^n than max^n and paranoid. This validates the results found in Table 2. Similar to the results found in Subsection 5.4, MCTS-max^n performs relatively better with higher time settings. The win rate of the minimax-based players drops as the players receive more thinking time. This is not only true for BRS, but also for max^n and paranoid. Similar to the experiments in Subsection 5.2, max^n does not apply normalization of the heuristic evaluation function and shallow pruning in Chinese Checkers in this set of experiments.

In Focus, the performance of the minimax-based techniques against MCTS-max^n decreases if more time is provided, as well. BRS wins approximately 60% of the games with a low time setting, but its win rate drops to between 30% and 35% with 5000 ms of thinking time. Paranoid is on equal footing with MCTS-max^n in three-player Focus with a low time setting, but if more time is provided, MCTS performs significantly better. Max^n wins less than 25% of the games against MCTS-max^n with any time setting and any number of players.

In Rolit, the three different minimax-based players win around or less than 30% of the games against MCTS-max^n. Paranoid wins slightly more games than BRS against MCTS-max^n, which validates that paranoid is at least as strong as BRS in Rolit. In Subsections 5.2 and 5.4 we found that both paranoid and MCTS-max^n perform at least as well as, or better than, BRS in Rolit. When comparing paranoid to MCTS-max^n, we find that the MCTS-based player performs best. Paranoid wins around or less than 30% of the games against MCTS-max^n with any number of players or time setting.

Finally, in Blokus, all minimax-based players are outperformed by MCTS-max^n for each time setting. In Subsection 5.2 we found that BRS is the strongest and max^n is the weakest minimax technique in Blokus. The results in Table 6 reveal a similar result.

Table 6: Win rates of the minimax-based techniques against MCTS-max^n.

Game	Players	Time (ms)	Max^n win rate (%)	Paranoid win rate (%)	BRS win rate (%)
Chinese Checkers	3	250	20.8±2.5	57.7±3.0	81.6±2.3
	3	1000	4.0±1.2	22.6±2.5	57.6±3.0
	3	5000	1.5±0.7	9.8±1.8	31.8±2.8
Chinese Checkers	4	250	33.2±2.8	21.3±2.5	75.5±2.6
	4	1000	6.7±1.5	12.6±2.0	42.3±3.0
	4	5000	3.0±1.0	3.9±1.2	22.4±2.5
Chinese Checkers	6	250	36.2±2.9	24.6±2.6	66.7±2.8
	6	1000	9.3±1.8	4.5±1.3	29.9±2.7
	6	5000	4.6±1.3	4.4±1.2	11.9±2.0
Focus	3	250	16.7±2.3	50.3±3.0	62.9±2.9
	3	1000	8.9±1.7	31.0±2.8	46.2±3.0
	3	5000	5.7±1.4	24.5±2.6	35.0±2.9
Focus	4	250	23.9±2.6	30.8±2.8	57.7±3.0
	4	1000	15.6±2.2	27.4±2.7	45.7±3.0
	4	5000	9.0±1.7	18.4±2.3	31.4±2.8
Rolit	3	250	9.2±1.7	31.4±2.8	25.9±2.6
	3	1000	5.4±1.4	20.7±2.5	15.4±2.2
	3	5000	4.4±1.2	16.7±2.3	13.0±2.0
Rolit	4	250	20.1±2.4	29.3±2.8	28.8±2.7
	4	1000	13.0±2.0	26.1±2.7	19.8±2.4
	4	5000	11.1±1.9	21.0±2.5	18.0±2.3
Blokus	4	250	23.5±2.6	32.4±2.8	42.2±3.0
	4	1000	5.9±1.4	10.6±1.9	22.6±2.5
	4	5000	1.2±0.7	2.1±0.9	9.5±1.8

General remarks

These experiments confirm the results found in Subsection 5.2. Max^n achieves the lowest win rate against MCTS-max^n, showing that max^n is the weakest minimax-based algorithm. The highest win rate is achieved by BRS, except in Rolit. In Rolit, paranoid has a slightly higher win percentage than BRS against the MCTS-max^n player, which is comparable to the results in Subsection 5.2, where we found that paranoid and BRS perform on a similar level. Furthermore, the results reveal that all three players perform worse against MCTS-max^n if more time is provided. A similar result was found in Subsection 5.4, where the performance of the MCTS-based search techniques increases against BRS if the amount of thinking time is increased.

6. BACKGROUND OF MCTS-SOLVER

The MCTS variants described in the previous sections are not able to solve positions. Winands *et al.* (2008) proposed a new MCTS variant for two-player games, called *MCTS-Solver*, which has been designed to play narrow tactical lines better in sudden-death games. A sudden-death game is a game that may end abruptly by the creation of one of a prespecified set of patterns (Allis, 1994). The variant differs from the traditional MCTS in respect to backpropagation and selection strategy. It is able to prove the game-theoretic value of a position given sufficient time.

In addition to backpropagating the values $\{0, \frac{1}{2}, 1\}$, representing a loss, a draw and a win respectively, the search also backpropagates the values ∞ and $-\infty$, which are assigned to a proven won or lost position, respectively. To prove that a node is a win, it is sufficient to prove that at least one of the children is a win. In order to prove that a node is a loss, it is necessary to prove that all children lead to a loss. If at least one of the children is not a proven loss, then the current node cannot be proven.

Experiments showed that for the sudden-death game Lines of Action (LOA), an MCTS program using MCTS-Solver defeats a program using MCTS by a winning percentage of 65%. Moreover, MCTS-Solver performs much better than MCTS against the world-class $\alpha\beta$ program MIA. They concluded that MCTS-Solver constitutes genuine progress in solving and playing strength in sudden-death games, significantly improving upon MCTS-based programs. The MCTS-Solver has also been successfully applied in games such as Hex (Arneson, Hayward, and Henderson, 2010; Cazenave and Saffidine, 2010), Shogi (Sato, Takahashi, and Grimbergen, 2010), and Tron (Den Teuling and Winands, 2012).

Cazenave and Saffidine (2011) proposed to improve the MCTS-Solver using *Score Bounded Monte-Carlo Tree Search* when a game has more than two outcomes, for example in games that can end in draw positions. It significantly improved the MCTS-Solver to take into account bounds on the possible scores of a node in order to select the nodes to explore. They applied this algorithm to solve Seki in the game of Go, and to Connect Four. Score Bounded Monte-Carlo Tree Search has also been applied in simultaneous move games (Finnsson, 2012).

7. MULTI-PLAYER MCTS-SOLVER

The previous experiments revealed that MCTS-max^n is the strongest multi-player MCTS variant. Therefore, for MCTS-max^n, we propose a multi-player variant of MCTS-Solver, called *Multi-Player MCTS-Solver* (MP-MCTS-Solver). For the multi-player variant, MCTS-Solver has to be modified, in order to accommodate for games with more than two players. This is discussed below.

Proving a win works similarly as in the two-player version of MCTS-Solver: if one of the children is a proven win for the player who has to move in the current node, then this node is a win for this player. If all children lead to a win for the same opponent, then the current node is also labelled as a win for this opponent. However, if the children lead to wins for different opponents, then updating the game-theoretic values becomes a non-trivial task.

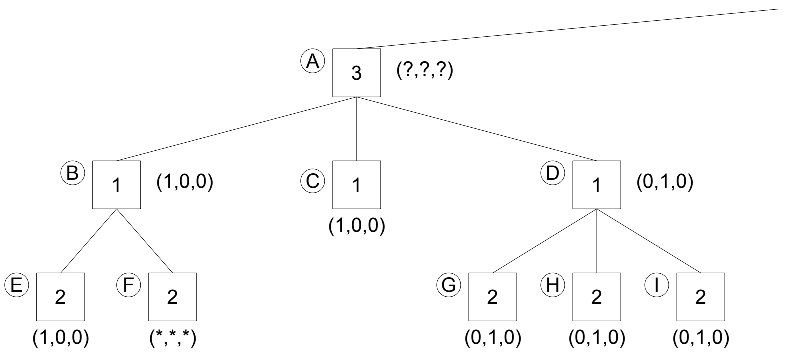


Figure 7: An example of backpropagating game-theoretic values in a multi-player search tree.

An example is given in Figure 7. Here, node E is a terminal node where Player 1 has won. This means that node B is a mate-in-one for Player 1, regardless of the value of node F. Node E is marked as solved and receives a game-theoretic value of $(1, 0, 0)$. Nodes G, H, and I all result in wins for Player 2. Parent node D receives a game-theoretic value of $(0, 1, 0)$, because this node always leads to a win for the same opponent of Player 1. The game-theoretic value of node A cannot be determined in this case, because both Player 1 and Player 2 are able to win and there is no win for Player 3.

Update rules have to be developed to take care of such situations. We propose three different update rules that are briefly explained below.

(1) The *normal* update rule only updates proven wins for the same opponent. This means that only if all children lead to a win for the same opponent, then the current node is also set to a win for this opponent. Otherwise, the node is not marked as solved and the UCT value is used. A disadvantage of the normal update rule is that it is quite conservative. We define two update rules that allow solving nodes that lead to different winners.

(2) The *paranoid* update rule uses the assumption that the opponents of the root player will never let him win. This rule is inspired by the paranoid tie breaker rule for max^n (Sturtevant, 2003a). Again consider Figure 7. Assuming that the root player is Player 1, using the paranoid update rule, we can determine the game-theoretic value of node A. Because we assume that Player 3 will not let Player 1 win, the game-theoretic value of node A becomes $(0, 1, 0)$. If there are still multiple winners after removing the root player from the list of possible winners, then no game-theoretic value is assigned to the node.

The paranoid update rule may not always give the desired result. With the paranoid assumption, the game-theoretic value of node A is $(0, 1, 0)$ (i.e., a win for Player 2). This is actually not certain, because it is also possible that Player 3 will let Player 1 win. However, because the game-theoretic value of node A denotes a win for Player 2, and at the parent of node A Player 2 is to move, the parent of node A will also receive a game-theoretic value of $(0, 1, 0)$. This is actually incorrect, since choosing node A does not give Player 2 a guaranteed win.

Problems may thus arise when a player in a given node gives the win to the player directly preceding him. In such a case, the parent node will receive a game-theoretic value which is technically not correct. This problem can be diminished by using (3) the *first-winner* update rule. When using this update rule, the player will give the win to the player who is the first winner after him. In this way the player before him does not get the win and, as a result, does not overestimate the position. When using the first-winner update rule, in Figure 7, node A will receive the game-theoretic value $(1, 0, 0)$.

Overestimation

Overestimation of a node is a phenomenon that occurs if one or more children of a node are proven to be a loss, but the node itself is not solved (yet). Winands, Björnsson, and Saito (2010) provided a case where overestimation may lead to wrong evaluations and showed how to tackle this problem by applying a threshold. If the number of visits in a node is less than the threshold, the simulation strategy is used to select a node. In this way, children that are proven to be a loss can be selected, as long as the number of visits is below the threshold. The UCT formula is applied to value solved children if a node is not proven. For the win rate $\bar{x}_i$, the game-theoretic value of the child is used, which is usually 0.[3] Overestimation is abated by occasionally selecting nodes that are a proven loss, but because the win rate is 0, non-proven nodes are favoured.

8. EXPERIMENTAL RESULTS FOR MULTI-PLAYER MCTS-SOLVER

In this section, we give the results of MP-MCTS-Solver with the three different update rules playing against an MCTS player without MP-MCTS-Solver. These experiments are only performed in Focus, because MCTS-Solver is only successful in sudden-death games (Winands *et al.*, 2008). Chinese Checkers, Rolit and Blokus do not belong to this category of games, and therefore MP-MCTS-Solver will not work well in these games. Focus, however, is a sudden-death game and is therefore an appropriate test domain for MP-MCTS-Solver.

Table 7: Win rates for an MP-MCTS-Solver player with different update rules against the default MCTS player in Focus.

Type	2 players win rate (%)	3 players win rate (%)	4 players win rate (%)
Standard	53.0±1.7	54.9±1.7	53.3±1.7
Paranoid	51.9±1.7	50.4±1.7	44.9±1.7
First-winner	52.8±1.7	51.5±1.7	43.4±1.7

In Table 7, we see that the standard update rule works well in Focus. The win rates for the different number of players vary between 53% and 55%. The other update rules do not perform as well. For the two-player variant, they behave and perform similar to the standard update rule. The win rates are slightly lower, which may be caused by statistical noise and a small amount of overhead. In the three-player variant, MP-MCTS-Solver neither increases nor decreases the performance significantly. In the four-player variant, the win rate of the player using MP-MCTS-Solver is well below 50% for the paranoid and the first-winner update rules. Based on these results we may conclude that only the standard update rule works well.

[3] If draws are allowed, the game-theoretic value may be non-zero. We remark that, in Focus, draws do not occur.

9. CONCLUSIONS AND FUTURE RESEARCH

Among the three minimax-based search techniques we tested, BRS turns out to be the strongest one. Overall, it reaches the highest search depth and, because of its tree structure, more MAX nodes are investigated than in paranoid and max^n. BRS significantly outperforms max^n and paranoid in Chinese Checkers, Focus and Blokus. Only in Rolit, paranoid performs at least as strong as BRS.

In the MCTS framework, the max^n search policy appears to perform best. The advantages of paranoid and BRS in the minimax framework do not apply in MCTS, because $\alpha\beta$ pruning is not applicable in MCTS. An additional problem with MCTS-BRS may be that, in the tree, invalid positions are investigated, which may reduce the reliability of the playouts as well. Still, MCTS-paranoid and MCTS-BRS overall achieve decent win rates against MCTS-max^n. Furthermore, MCTS-paranoid is on equal footing with MCTS-max^n in Blokus and, in the vanilla version of MCTS, MCTS-paranoid and MCTS-BRS significantly outperform MCTS-max^n in Focus. Although the max^n search policy is the most robust, the BRS and paranoid search policies can still be competitive.

In a comparison between MCTS-max^n and BRS, MCTS-max^n overall wins more games than BRS. In Chinese Checkers and Focus, BRS is considerably stronger with lower time settings, while in Rolit and Blokus MCTS-max^n significantly outperforms BRS. With higher time settings, MCTS-max^n outperforms BRS in all games with any number of players. From this we may conclude that with higher time settings, the MCTS-based player performs relatively better.

Finally, we proposed MP-MCTS-Solver in MCTS-max^n with three different update rules, namely (1) standard, (2) paranoid and (3) first-winner. This variant is able to prove the game-theoretic value of a position. We tested this variant only in Focus, because MP-MCTS-Solver only works well in sudden-death games. A win rate between 53% and 55% was achieved in Focus with the standard update rule. The other two update rules achieved a win rate up to 53% in the two-player variant, but were around or below 50% for the three- and four-player variants. We may conclude that MP-MCTS-Solver performs well with the standard update rule. The other two update rules, paranoid and first-winner, were not successful in Focus.

In this article we investigated three common search policies for multi-player games, i.e. max^n, paranoid and BRS, in the MCTS framework. We did not consider algorithms derived from these techniques, such as the Coalition-Mixer (Lorenz and Tscheuschner, 2006) or MP-Mix (Zuckerman, Felner, and Kraus, 2009). They use a combination of max^n and (variations of) paranoid search. They also have numerous parameters that have to be optimized. Tuning and testing such algorithms for multi-player MCTS is a first direction of future research.

A second possible future research direction is the application of BRS variants as proposed by Esser (2012) and Gras (2012) in MCTS. The basic idea is that, beside the opponent with the best counter move, the other opponents are also allowed to perform a move. These moves are selected using static move ordering. The advantage of these variants is that no invalid positions are searched, while maintaining the advantages of the original BRS algorithm.

A third future research topic is the application of paranoid and BRS policies in the playouts of MCTS. Cazenave (2008) applied paranoid playouts to multi-player Go and found promising results. BRS may be able to shorten the playouts, because all but one opponents skip their turn. This may increase the number of playouts per second, and thus increase the playing strength. Applying paranoid and BRS playouts requires developing and implementing paranoid moves for the opponents.

MP-MCTS-Solver has proven to be a genuine improvement for the sudden-death game Focus, though more research is necessary to improve its performance. As a fourth direction of future research, it may be interesting to investigate the performance of MP-MCTS-Solver in different sudden-death games. Furthermore, MP-MCTS-Solver is currently only applied in MCTS-max^n. It may be interesting to apply it to MCTS-paranoid and MCTS-BRS as well.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees of this article, and of the authors' contributions to the Computer Games Workshop at ECAI 2012 and the Computers and Games 2010 conference, for their useful suggestions and constructive criticism. Furthermore, the authors would like to thank Marc Lanctot for proof reading this article and his suggestions for improving its quality.

10. REFERENCES

Akl, S. G. and Newborn, M. M. (1977). The Principal Continuation and the Killer Heuristic. *Proceedings of the ACM Annual Conference*, pp. 466–473, ACM, New York, NY, USA.

Allis, L. V. (1994). *Searching for Solutions in Games and Artificial Intelligence.* Ph.D. thesis, Rijksuniversiteit Limburg, Maastricht, The Netherlands.

Arneson, B., Hayward, R. B., and Henderson, P. (2010). Monte-Carlo Tree Search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 251–258.

Breuker, D. M., Uiterwijk, J. W. H. M., and Herik, H. J. van den (1996). Replacement Schemes and Two-Level Tables. *ICCA Journal*, Vol. 19, No. 3, pp. 175–180.

Cazenave, T. (2008). Multi-player Go. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *LNCS*, pp. 50–59, Springer-Verlag, Berlin, Germany.

Cazenave, T. and Saffidine, A. (2010). Monte-Carlo Hex. *Board Game Studies XIIIth Colloquium*, Paris, France.

Cazenave, T. and Saffidine, A. (2011). Score Bounded Monte-Carlo Tree Search. *Computers and Games (CG 2010)* (eds. H. J. van den Herik, H. Iida, and A. Plaat), Vol. 6515 of *LNCS*, pp. 93–104, Springer-Verlag, Berlin, Germany.

Chaslot, G. M. J.-B., Winands, M. H. M., Uiterwijk, J. W. H. M., Herik, H. J. van den, and Bouzy, B. (2008). Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, Vol. 4, No. 3, pp. 343–357.

Clune, J. E. (2007). Heuristic Evaluation Functions for General Game Playing. *Proceedings of the Twenty-Second AAAI on Artificial Intelligence*, Vol. 22, pp. 1134–1139, AAAI Press.

Coulom, R. (2007). Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *Computers and Games (CG 2006)* (eds. H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers), Vol. 4630 of *LNCS*, pp. 72–83, Springer-Verlag, Berlin, Germany.

Den Teuling, N. G. P. and Winands, M. H. M. (2012). Monte-Carlo Tree Search for the Simultaneous Move Game Tron. *Computer Games Workshop at the ECAI 2012*, pp. 126–141, Montpellier, France.

Esser, M. (2012). Best-Reply Search in Multi-Player Chess. M.Sc. thesis, Maastricht University, Maastricht, The Netherlands.

Finnsson, H. (2012). *Simulation-Based General Game Playing.* Ph.D. thesis, School of Computer Science, Reykjavik University, Reykjavik, Iceland.

Gras, M. (2012). Multi-Player Search in the Game of Billabong. M.Sc. thesis, Maastricht University, Maastricht, The Netherlands.

Greenblatt, R. D., Eastlake, D. E., and Crocker, S. D. (1967). The Greenblatt Chess Program. *Proceedings of the AFIPS '67 Fall Joint Computer Conference*, Vol. 31, pp. 801–810.

Kocsis, L. and Szepesvári, C. (2006). Bandit Based Monte-Carlo Planning. *Machine Learning: ECML 2006* (eds. J. Fürnkranz, T. Scheffer, and M. Spiliopoulou), Vol. 4212 of *LNCS*, pp. 282–293, Springer-Verlag, Berlin, Germany.

Korf, R. E. (1991). Multi-Player Alpha-Beta Pruning. *Artificial Intelligence*, Vol. 48, No. 1, pp. 99–111.

Lorenz, U. and Tscheuschner, T. (2006). Player Modeling, Search Algorithms and Strategies in Multi-player Games. *Advances in Computer Games (ACG 11)* (eds. H. J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H. H. L. M. Donkers), Vol. 4250 of *LNCS*, pp. 210–224, Springer-Verlag, Berlin, Germany.

Luckhardt, C. A. and Irani, K. B. (1986). An Algorithmic Solution of N-Person Games. *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI)* (ed. T. Kehler), Vol. 1, pp. 158–162, Morgan Kaufmann.

Neumann, J. von and Morgenstern, O. (1944). *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, NJ, USA, second edition.

Nijssen, J. A. M. and Winands, M. H. M. (2011). Enhancements for Multi-Player Monte-Carlo Tree Search. *Computers and Games (CG 2010)* (eds. H. J. van den Herik, H. Iida, and A. Plaat), Vol. 6515 of *LNCS*, pp. 238–249, Springer-Verlag, Berlin, Germany.

Nijssen, J. A. M. and Winands, M. H. M. (2012). An Overview of Search Techniques in Multi-Player Games. *Computer Games Workshop at the ECAI 2012*, pp. 50–61, Montpellier, France.

Sackson, S. (1969). *A Gamut of Games.* Random House, New York, NY, USA.

Saito, J.-T. and Winands, M. H. M. (2010). Paranoid Proof-Number Search. *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games* (eds. G. N. Yannakakis and J. Togelius), pp. 203–210, IEEE.

Sato, Y., Takahashi, D., and Grimbergen, R. (2010). A Shogi Program Based on Monte-Carlo Tree Search. *ICGA Journal*, Vol. 33, No. 2, pp. 80–92.

Schadd, M. P. D. and Winands, M. H. M. (2011). Best Reply Search for Multiplayer Games. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 3, No. 1, pp. 57–66.

Schaeffer, J. (1983). The History Heuristic. *ICCA Journal*, Vol. 6, No. 3, pp. 16–19.

Shibahara, K. and Kotani, Y. (2008). Combining Final Score with Winning Percentage by Sigmoid Function in Monte-Carlo Simulations. *Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games* (eds. P. Hingston and L. Barone), pp. 183–190, IEEE.

Sturtevant, N. R. (2003a). A Comparison of Algorithms for Multi-Player Games. *Computers and Games (CG 2002)* (eds. J. Schaeffer, M. Müller, and Y. Björnsson), Vol. 2883 of *LNCS*, pp. 108–122, Springer-Verlag, Berlin, Germany.

Sturtevant, N. R. (2003b). Last-Branch and Speculative Pruning Algorithms for Max^n. *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence* (eds. G. Gottlob and T. Walsh), pp. 669–675, Morgan Kaufmann.

Sturtevant, N. R. (2008). An Analysis of UCT in Multi-player Games. *ICGA Journal*, Vol. 31, No. 4, pp. 195–208.

Sturtevant, N. R. and Korf, R. E. (2000). On Pruning Techniques for Multi-Player Games. *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pp. 201–207, AAAI Press / The MIT Press.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, USA.

Tak, M. J. W., Winands, M. H. M., and Björnsson, Y. (2012). N-Grams and the Last-Good-Reply Policy applied in General Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 4, No. 2, pp. 73–83.

Winands, M. H. M., Björnsson, Y., and Saito, J.-T. (2008). Monte-Carlo Tree Search Solver. *Computers and Games (CG 2008)* (eds. H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands), Vol. 5131 of *LNCS*, pp. 25–36, Springer-Verlag, Berlin, Germany.

Winands, M. H. M., Björnsson, Y., and Saito, J.-T. (2010). Monte Carlo Tree Search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 2, No. 4, pp. 239–250.

Zuckerman, I., Felner, A., and Kraus, S. (2009). Mixing Search Strategies for Multi-Player Games. *Proceedings of the Twenty-first International Joint Conferences on Artificial Intelligence (IJCAI-09)* (ed. C. Boutilier), pp. 646–651, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.