Playout Search for Monte-Carlo Tree Search in Multi-Player Games

J. (Pim) A.M. Nijssen and Mark H.M. Winands

Games and AI Group, Department of Knowledge Engineering,
Faculty of Humanities and Sciences,
Maastricht University, Maastricht, The Netherlands
{pim.nijssen,m.winands}@maastrichtuniversity.nl

Abstract. Monte-Carlo Tree Search (MCTS) has become a popular search technique for playing multi-player games over the past few years. In this paper we propose a technique called Playout Search. This enhancement allows the use of small searches in the playout phase of MCTS in order to improve the reliability of the playouts. We investigate $\max^n$, Paranoid and BRS for Playout Search and analyze their performance in two deterministic perfect-information multi-player games: Focus and Chinese Checkers. The experimental results show that Playout Search significantly increases the quality of the playouts in both games. However, it slows down the speed of the playouts, which outweighs the benefit of better playouts if the thinking time for the players is small. When the players are given a sufficient amount of thinking time, Playout Search employing Paranoid search is a significant improvement in the 4-player variant of Focus and the 3-player variant of Chinese Checkers.

1 Introduction

Deterministic perfect-information multi-player games pose an interesting challenge for computers. In the past the standard techniques to play these games were $\max^n$ [13] and Paranoid [20]. Similar to for instance Best Reply Search (BRS) [18] and Coalition-Mixer [12], these search techniques use an evaluation function to determine the values of the leaf nodes in the tree. Applying search is generally more difficult in multi-player games than in 2-player games. Pruning in the game tree of a multi-player game is much harder [19]. With $\alpha\beta$ pruning, the size of a tree in a 2-player game can be reduced from $O(b^d)$ to $O(b^{\frac{d}{2}})$ in the best case. In Paranoid, the size of the game tree can only be reduced to $O(b^{\frac{n-1}{n}d})$ in the best case and in BRS, the size can be reduced to $O\left((b(n-1))^{\lceil \frac{2d}{n} \rceil/2}\right)$. When using $\max^n$, safe pruning is hardly possible. Also, opponent's moves are less predictable. Contrary to 2-player games, where two players always play against each other, in multi-player games (temporary) coalitions might occur. This can change the behavior of the opponents.

Over the past years, Monte-Carlo Tree Search (MCTS) [7, 10] has become a popular technique for playing multi-player games. MCTS is a best-first search

technique that instead of an evaluation function uses simulations to guide the search. Next, MCTS is able to compute mixed equilibria in multi-player games [19], contrary to max^n, Paranoid and BRS. MCTS is used in a variety of multi-player games, such as Focus [15], Chinese Checkers [15, 19], Hearts [19], Spades [19], and multi-player Go [5].

For MCTS, a tradeoff between search and knowledge has to be made. The more knowledge is added, the slower each playout gets. The trend seems to favor fast simulations with computationally light knowledge, although recently, adding more heuristic knowledge at the cost of slowing down the playouts has proven beneficial in some games [21]. Game-independent enhancements in the playout phase of MCTS such as Gibbs sampling [2] and RAVE [16] have proven to increase the playing strength of MCTS programs significantly. With ϵ-greedy playouts [19], some simple game-specific knowledge can be incorporated. Lorentz [11] improved the playing strength of the MCTS-based Havannah program WANDERER by checking whether the opponent has a 'mate-in-one' when selecting a move in the beginning of the playout. Winands and Björnsson [21] proposed $\alpha\beta$-based playouts for the 2-player game Lines of Action. Although computationally intensive, it significantly improved the playing strength of the MCTS program.

In this paper we propose Playout Search for MCTS in multi-player games. Instead of using computationally light knowledge in the playout phase, small two-ply searches are used to determine the moves to play. We test three different search techniques that may be used for Playout Search. These search techniques are max^n, Paranoid and BRS. Playout Search is tested in two disparate multi-player games: Focus and Chinese Checkers.

The remainder of the paper is structured as follows. First, Section 2 gives a brief overview of the application of MCTS in multi-player games. Next, Playout Search is introduced in Section 3. An overview of the rules and domain knowledge for Focus and Chinese Checkers is given in Section 4. Subsequently, Section 5 describes the experiments and the results. Finally, the conclusions and an outlook on future research are given in Section 6.

2 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) [7, 10] is a search technique that gradually builds up a search tree, guided by Monte-Carlo simulations. In contrast to classic search techniques such as $\alpha\beta$-search [9], it does not require a heuristic evaluation function. The MCTS algorithm consists of four phases [6]: selection, expansion, playout and backpropagation (see Fig. 1). By repeating these four phases iteratively, the search tree is constructed gradually. Below we explain the application to multi-player games for our MCTS program [15].

In the *selection* phase the search tree is traversed from the root node until a node is found that contains children that have not been added to the tree yet. The tree is traversed using the Upper Confidence bounds applied to Trees (UCT) [10] selection strategy. In our program, we have enhanced UCT with Progressive History [15]. The child i with the highest score v_i in Formula 1 is selected.

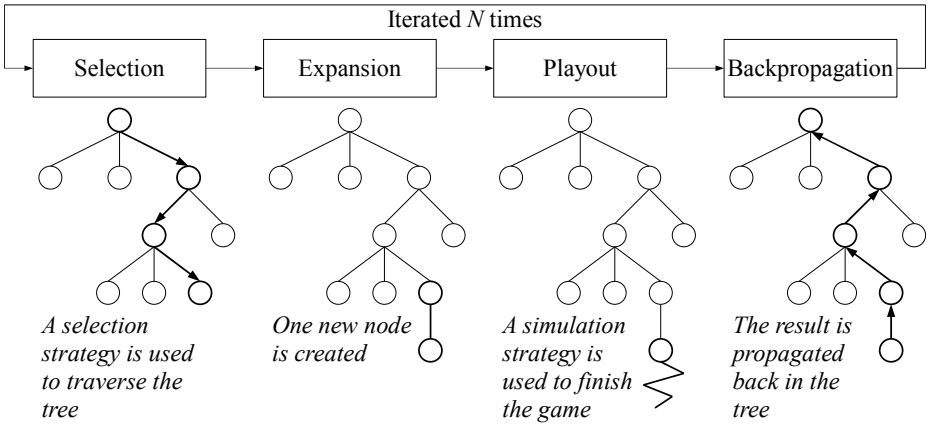


Fig. 1. Monte-Carlo Tree Search scheme (Slightly adapted from [6]).

$$v_i = \frac{s_i}{n_i} + C \times \sqrt{\frac{\ln(n_p)}{n_i}} + \frac{s_a}{n_a} \times \frac{W}{n_i - s_i + 1} \tag{1}$$

In this formula, s_i denotes the total score of child i, where a win is being rewarded 1 point and a loss 0 points. The variables n_i and n_p denote the total number of times that child i and parent p have been visited, respectively. C is a constant, which determines the exploration factor of UCT. In the Progressive History part, s_a represents the score of move a, where each playout in which a was played resulted in a win adds 1 point and a loss 0 points. n_a is the number of times move a has been played in any previous playout. W is a constant that determines the influence of Progressive History.

In the *expansion* phase one node is added to the tree. Whenever a node is found which has children that have not been added to the tree yet, then one of these children is chosen and added to the tree [7].

During the *playout* phase, moves are played in self-play until the game is finished. Usually, the playouts are being generated using random move selection. However, progression has been identified as an important success factor for MCTS [8, 22]. Ideally, each move should bring the game closer towards its conclusion. Otherwise, there is a risk of the simulations leading mostly to futile results. In slow-progressing games, such as Chinese Checkers and Focus (see Section 4), knowledge should be added to the playouts [3] to ensure a quick resolution of the game. Often, simple evaluations are used to select the moves to play. In our MCTS program, the following two strategies have been incorporated. 1) When using a *move evaluator*, a heuristic is used to assign a value to all valid moves of the current player. The move with the highest evaluation score is chosen. The move evaluator is fast, but it only considers a local area of the board. 2) With *one-ply search*, all valid moves of the current player are performed and the resulting board positions are evaluated. The move which gives the best board

position, i.e., the highest evaluation score for the current player, is chosen. The board evaluator is slower than the move evaluator, but it gives a more global evaluation. Knowledge can also be incorporated by employing 2-ply searches to determine the move to play. In Section 3 we explain which search techniques are used.

Finally, in the *backpropagation* phase, the result is propagated back along the previously traversed path up to the root node. In the multi-player variant of MCTS, the result is a tuple of size N, where N is the number of players. The value corresponding to the winning player is 1, the value corresponding to the other players is 0. The game-theoretic values of terminal nodes are stored and, if possible, backpropagated in such a way that MCTS is able to prove a (sub)tree [15, 22].

This four-phase process is repeated either a fixed number of times, or until the time is up. When the process is finished, the child of the root node with the highest win rate is returned.

3 Playout Search

In this section we propose Playout Search for MCTS in multi-player games. In Subsection 3.1 we explain which search techniques are used in the playout phase. In Subsection 3.2 we describe which enhancements are used to speed up the search.

3.1 Search Techniques

Instead of playing random moves biased by computationally light knowledge in the playout phase, domain knowledge can be incorporated by performing small searches. This reduces the number of playouts per second significantly, but it improves the reliability of the playouts. When selecting a move in the playout phase, one of the following three search techniques is used to choose a move.

1) Two-ply max^n [13]. A two-ply max^n search tree is built where the current player is the root player and the first opponent plays at the second ply. Both the root player and the first opponent try to maximize their own score. $\alpha\beta$-pruning in a two-ply max^n search tree is not possible.

2) Two-ply Paranoid [20]. Similar to max^n, a two-ply search tree is built where the current player is the root player and the first opponent plays at the second ply. The root player tries to maximize its own score, while the first opponent tries to minimize the root player's score. In a two-ply Paranoid search tree, $\alpha\beta$-pruning is possible.

3) Two-ply Best Reply Search (BRS) [18]. BRS is similar to Paranoid search. The difference is that at the second ply, not only the moves of the first opponent are considered, but the moves of all opponents are investigated. Similar to Paranoid search, $\alpha\beta$-pruning is possible.

3.2 Search Enhancements

The major disadvantage of incorporating search in the playout phase of MCTS is the reduction of the number of playouts per second [21]. In order to prevent this reduction from outweighing the benefit of the quality of the playouts, enhancements should be implemented to speed up the search and keep the reduction of the number of playouts to a minimum. In our MCTS program, the following enhancements to speed up the playout search are used.

The number of searches can be reduced by using *ϵ-greedy playouts* [19]. With a probability of ϵ, a move is chosen uniform randomly. Otherwise, the selected search technique is used to select the best move. An additional advantage of ϵ-greedy playouts is that the presence of this random factor gives more varied playouts and prevents the playouts from being stuck in 'local optima', where all players keep moving back and forth. ϵ-greedy playouts are used with all aforementioned playout strategies.

The amount of $\alpha\beta$-pruning in a tree can be increased by using *move ordering*. When using move ordering, a player's moves are sorted using a static move evaluator. In the best case, the number of evaluated board positions in a two-ply search is reduced from b^2 to $2b - 1$ [9]. The size of the tree can be further reduced by using *k-best pruning*. Only the k best moves are investigated. This reduces the branching factor of the tree from b to k. The parameter k should be chosen such that it is significantly smaller than b, while avoiding the best move being pruned. Move ordering and k-best pruning are used in all techniques described in Subsection 3.1.

Another move ordering technique is applying *killer moves* [1]. In each search, two killer moves are always tried first. These are the two last moves that were best or caused a cutoff, at the current depth. Moreover, if the search is completed, the killer moves for that specific level in the playout are stored, such that they can be used during the next MCTS iterations. Killer moves are only used with search techniques where $\alpha\beta$-pruning is possible, i.e., Paranoid and BRS search.

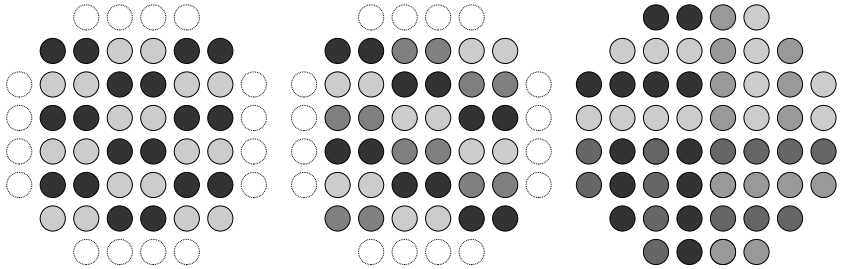
Other enhancements were tested, but they did not improve the performance of the MCTS program. The application of *transposition tables* [4] was tested, but the information gain did not compensate for the overhead. Also, *aspiration search* [14] did not speed up the search significantly. This can be attributed to the limited amount of pruning possible in a two-ply search tree.

4 Test Domains

Playout Search is tested in two different games: Focus and Chinese Checkers. In this section we briefly discuss the rules and the properties of Focus and Chinese Checkers in Subsection 4.1 and 4.2, respectively. In Subsection 4.3 we explain the move and board evaluators for Focus and Chinese Checkers.

4.1 Focus

Focus is an abstract multi-player strategy board game, which was invented in 1963 by Sid Sackson [17]. This game has also been released under the name



(a) 2 players (b) 3 players (c) 4 players

Fig. 2. Set-ups for Focus

Domination. Focus is played on an 8×8 board where in each corner three fields are removed. It can be played by 2, 3 or 4 players. Each player starts with a number of pieces on the board. In Fig. 2, the initial board positions for the 2-, 3- and 4-player variants are given.

In Focus, pieces can be stacked on top of each other. A stack may contain up to 5 pieces. Each turn a player may move a stack orthogonally as many fields as the stack is tall. A player may only move a stack of pieces if a piece of his color is on top of the stack. It is also allowed to split stacks in two smaller stacks. If a player decides to do so, then he only moves the upper stack as many fields as the number of pieces that are being moved.

If a stack lands on top of another stack, then the stacks are merged. If the merged stack has a size of $n > 5$, then the bottom $n - 5$ pieces are captured by the player, such that there are 5 pieces left. If a player captures one of his own pieces, he may later choose to place one piece back on the board, instead of moving a stack. This piece may be placed either on an empty field or on top of an existing stack.

There exist two variations of the game, each with a different winning condition. In the standard version of the game, a player has won if all other players cannot make a legal move. However, such games can take a long time to finish. Therefore, we chose to use the shortened version of the game. In this version, a player has won if he has either captured certain number of pieces in total, or a number of pieces from each player. In the 2-player variant, a player wins if he has captured at least 6 pieces from the opponent. In the 3-player variant, a player has won if he has captured at least 3 pieces from both opponents or at least 10 pieces in total. In the 4-player variant, the goal is to capture at least 2 pieces from each opponent or capture at least 10 pieces in total.

4.2 Chinese Checkers

Chinese Checkers is a board game that can be played by 2 to 6 players. This game was invented in 1893 and has since then been released by various publishers

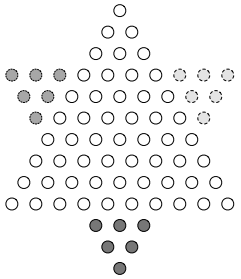


Fig. 3. A Chinese Checkers board [19].

under different names. Chinese Checkers is played on a star-shaped board. The most commonly used board contains 121 fields, where each player starts with 10 checkers. We decided to play on a slightly smaller board [19] (see Fig. 3). In this version, each player plays with 6 checkers. The advantage of a smaller board is that games take a shorter amount of time to complete, which means that more Monte-Carlo simulations can be performed and more experiments can be run. Also, it allows the use of a stronger evaluation function.

The goal of each player is to move all his pieces to his home base at the other side of the board. Pieces may move to one of the adjacent fields or they may jump over another piece to an empty field. It is also allowed to make multiple jumps with one piece in one turn, making it possible to create a setup that allows pieces to jump over a large distance. The first player who manages to fill his home base wins the game.

4.3 Domain Knowledge

For Chinese Checkers, the value of a move equals $d_s - d_t$, where d_s is the distance of the source location of the piece that is moved to the home base, and d_t the distance of the target location to the home base. For each location on the board, the distance to each home base is stored in a table. Note that the value of a move is negative if the piece moves away from the home base. For determining the board value, a lookup table [19] is used. This table stores, for each possible configuration of pieces, the minimum number of moves a player should perform to get all pieces in the home base, assuming that there are no opponents' pieces on the board. For any player, the value of a board equals $28 - m$, where m is the value stored in the table which corresponds to the configuration of the pieces of the player. Note that 28 is the highest value stored in the table.

For Focus, the value of a move equals $10(n + t) + s$, where n is the number of pieces moved, t is the number of pieces on the target location, and s is the number of stacks the player gained. The value of s can be 1, 0, or –1. For any player, the board value is based on the minimum number of pieces the player needs to capture to win the game, r, and the number of stacks the player controls, c. The score is calculated using the formula $600 - 100r + c$.

Table 1. 95% confidence intervals of some winning rates for 1500 games.

Win percentage	Confidence interval
50%	± 2.5%
40% / 60%	± 2.5%
30% / 70%	± 2.3%
20% / 80%	± 2.0%

5 Experiments

In this section, we describe the experiments that were performed to investigate the strength of Playout Search for MCTS in Focus and Chinese Checkers. In Subsection 5.1 the experimental setup is given. In Subsection 5.2 we present the experimental results for the different search techniques of Playout Search in Focus and Chinese Checkers.

5.1 Experimental Setup

The MCTS engines of Focus and Chinese Checkers are written in Java [15]. For Formula 1, the constant C is set to 0.2 and W is set to 5. All players use ϵ-greedy playouts with $\epsilon = 0.05$. The value of k for k-best pruning is set to 5. These values were achieved by systematic testing. The experiments were run on a cluster containing of AMD64 Opteron 2.4 GHz processors. In order to test the performance of Playout Search, we performed several round-robin tournaments where each participating player uses a different playout strategy. These playout strategies include 2-ply $\max^n$ (M), 2-ply Paranoid (P) and 2-ply BRS (B). Additionally, we include players with one-ply (O) and move evaluator (E) playouts as reference players. The tournaments were run for 3-player and 4-player Chinese Checkers and 3-player and 4-player Focus. In each game, two different player types participate. If one player wins, a score of 1 is added to the total score of the corresponding player type. For both games, there may be an advantage regarding the order of play and the number of different players. In a 3-player game there are $2^3 = 8$ different player-type assignments. Games where only one player type is playing are not interesting, leaving 6 ways to assign player types. For four players, there are $2^4-2 = 14$ assignments. Each assignment is played multiple times until approximately 1,500 games are played and each assignment was played equally often. In Table 1, 95% confidence intervals of some winning rates for 1500 games are given.

5.2 Results

In the first set of experiments, all players were allowed to perform 5000 playouts per move. The results are given in Table 2. The numbers are the win percentages of the players denoted on the left against the players denoted at the top.

The results show that for 3-player Chinese Checkers, BRS is the best technique. It performs slightly better than $\max^n$ and Paranoid. BRS wins 53.4% of

Table 2. Round-robin tournament of the different search techniques in Chinese Checkers and Focus with 5000 playouts per move (win%).

	E	O	M	P	B	Avg.
Move eval.	-	25.2	20.9	21.2	18.3	21.4
One-ply	74.8	-	44.5	40.5	38.9	49.7
Max^n	79.1	55.5	-	48.1	46.6	57.3
Paranoid	78.8	59.5	51.9	-	49.1	59.8
BRS	81.7	61.1	53.4	50.9	-	61.8

3-player Chinese Checkers

	E	O	M	P	B	Avg.
Move eval.	-	44.5	38.4	38.5	33.3	38.7
One-ply	55.5	-	44.3	44.1	40.5	46.1
Max^n	61.6	55.7	-	52.0	45.2	53.6
Paranoid	61.5	55.9	48.0	-	44.5	52.5
BRS	66.7	59.5	54.8	55.5	-	59.1

3-player Focus

	E	O	M	P	B	Avg.
Move eval.	-	30.3	27.6	26.9	22.9	26.9
One-ply	69.7	-	47.4	45.1	39.7	50.5
Max^n	72.4	52.6	-	49.1	48.1	55.6
Paranoid	73.1	54.9	50.9	-	46.2	56.3
BRS	77.1	60.3	51.9	53.8	-	60.8

4-player Chinese Checkers

	E	O	M	P	B	Avg.
Move eval.	-	42.0	35.0	35.2	33.4	36.4
One-ply	58.0	-	43.3	42.6	40.1	46.0
Max^n	65.0	56.7	-	50.5	48.5	55.2
Paranoid	64.8	57.4	49.5	-	48.2	55.0
BRS	66.6	59.9	51.5	51.8	-	57.5

4-player Focus

Table 3. Playouts per second for each type of player in each game variant.

Game	Move eval.	One-ply	Max^n	Paranoid	BRS
3-player Focus	7003	6138	2336	3356	1911
4-player Focus	6976	6237	2344	3410	1887
3-player Chinese Checkers	7322	6047	3439	4307	3890
4-player Chinese Checkers	5818	4630	2407	3066	2536

the games against max^n and 50.9% against Paranoid. These three techniques perform significantly better than one-ply and the move evaluator. The win rates against one-ply vary from 55.5% to 61.6% and against the move evaluator from 78.8% to 81.7%. In the 4-player variant, max^n, Paranoid and BRS remain the best techniques, where BRS performs slightly better than the other two. BRS wins 53.8% of the games against Paranoid and 51.9% against max^n. The win rates of max^n, Paranoid and BRS vary from 72.4% to 77.1% against the move evaluator and from 52.6% to 60.3% against one-ply.

For 3-player Focus, the best technique is BRS, winning 54.8% against max^n and 55.5% against Paranoid. Max^n and Paranoid are equally strong. The win rates of max^n, Paranoid and BRS vary between 61.5% and 66.7% against the move evaluator and between 55.7% and 59.5% against one-ply. BRS is also the best technique in 4-player Focus, though it is closely followed by max^n and Paranoid. BRS wins 51.5% of the games against max^n and 51.8% against Paranoid.

In the second set of experiments, we gave each player 5 seconds per move. For reference, Table 3 shows the average number of playouts per second for each type of player in each game variant. Note that at the start of the game, the number of playouts is smaller. As the game progresses, the playouts become shorter and the number of playouts per second increases.

The results of the round-robin tournament are given in Table 4. In 3-player Chinese Checkers, one-ply and Paranoid are the best techniques. Paranoid wins

Table 4. Round-robin tournament of the different search techniques in Chinese Checkers and Focus for time settings of 5 seconds per move (win%).

	E	O	M	P	B	Avg.
Move eval.	-	28.7	42.7	31.5	36.1	34.8
One-ply	71.3	-	62.5	50.8	58.2	60.7
Max^n	57.3	37.5	-	36.1	43.5	43.5
Paranoid	68.5	49.2	63.9	-	55.7	59.3
BRS	63.9	41.8	56.5	44.3	-	51.6

3-player Chinese Checkers

	E	O	M	P	B	Avg.
Move eval.	-	43.2	54.2	48.1	51.8	49.3
One-ply	56.8	-	58.9	53.9	57.9	56.9
Max^n	45.8	41.1	-	43.5	50.7	45.3
Paranoid	51.9	46.1	56.5	-	52.7	51.8
BRS	48.2	42.1	49.3	47.3	-	46.7

3-player Focus

	E	O	M	P	B	Avg.
Move eval.	-	33.7	45.9	35.4	42.9	39.5
One-ply	66.3	-	60.5	53.7	56.2	59.2
Max^n	54.1	39.5	-	40.3	46.6	45.1
Paranoid	64.6	46.3	59.7	-	56.2	56.7
BRS	57.1	43.8	53.4	43.8	-	49.5

4-player Chinese Checkers

	E	O	M	P	B	Avg.
Move eval.	-	42.3	41.1	40.1	43.9	49.1
One-ply	57.7	-	51.3	48.3	54.5	53.0
Max^n	58.9	48.7	-	47.9	55.9	52.9
Paranoid	59.9	51.7	52.1	-	54.3	54.5
BRS	56.1	45.5	44.1	45.7	-	47.9

4-player Focus

Table 5. Win rates of the Paranoid player against the one-ply player for time settings of 5 and 30 seconds per move.

Game	5 seconds	30 seconds
3-player Chinese Checkers	49.2%	53.9%
4-player Chinese Checkers	46.3%	48.3%
3-player Focus	46.1%	50.7%
4-player Focus	51.7%	54.1%

49.2% of the games against one-ply and 68.5% against the move evaluator. BRS ranks third, and the move evaluator and max^n are the weakest techniques. In 4-player Chinese Checkers, one-ply is the best technique, closely followed by Paranoid. One-ply wins 53.7% of the games against Paranoid. Paranoid is still stronger than the move evaluator, winning 64.6% of the games. BRS comes in third place, outperforming max^n and the move evaluator.

One-ply also performs the best in 3-player Focus. Paranoid plays slightly stronger than the move evaluator, with Paranoid winning 51.9% of the games against the move evaluator. One-ply wins 56.8% of the games against the move evaluator and 53.9% against Paranoid. The move evaluator and Paranoid perform better than BRS and max^n. In 4-player Focus, Paranoid performs better than in the 3-player version and outperforms one-ply. Paranoid wins 51.7% of the games against one-ply and 59.9% against the move evaluator. Max^n also performs significantly better than in the 3-player version. It is as strong as one-ply and better than the move evaluator, winning 58.9% of the games.

In the final set of experiments, we gave the players 30 seconds per move. Because these games take quite some time to finish, only the one-ply player and the Paranoid player were matched against each other. In the previous set of experiments, these two techniques turned out to be the strongest. The results are given in Table 5.

Paranoid appears to perform slightly better when the players receive 30 seconds per move compared to 5 seconds per move. In 3-player Chinese Checkers, Paranoid wins 53.9% of the games, compared to 49.2% with 5 seconds. In 4-player Chinese Checkers, 48.3% of the games are won by Paranoid, compared to 46.3% with 5 seconds. In 3-player Focus, the win rate of Paranoid increases from 46.1% with 5 seconds to 50.7% with 30 seconds and in 4-player Focus from 51.7% to 54.1%.

6 Conclusions and Future Research

In this paper we proposed Playout Search for improving the playout phase of MCTS in multi-player games. We applied 2-ply $\max^n$, Paranoid and BRS searches to select the moves to play in the playout phase. Some enhancements, such as ϵ-greedy playouts, move ordering, killer moves and k-best pruning were implemented to speed up the search.

The results show that Playout Search significantly improves the quality of the playouts in MCTS. This benefit is countered by a reduction of the number of playouts per second. Especially BRS and $\max^n$ suffer from this effect. Based on the experimental results we may conclude that Playout Search for multi-player games might be beneficial if the players receive sufficient thinking time and Paranoid search is employed. Under these conditions, Playout Search outperforms playouts using light heuristic knowledge in the 4-player variant of Focus and the 3-player variant of Chinese Checkers.

There are two directions for future research. First, it may be interesting to test Playout Search in other games as well. Second, the two-ply searches may be further optimized. Though a two-ply search will always be slower than a one-ply search, the current speed difference could be reduced further. This can be achieved for instance by improved move ordering or lazy evaluation functions.

References

1. S.G. Akl and M.M. Newborn. The Principal Continuation and the Killer Heuristic. In *Proceedings of the ACM Annual Conference*, pages 466–473, New York, NY, USA, 1977. ACM.
2. Y. Björnsson and H. Finnsson. CadiaPlayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1):4–15, 2009.
3. B. Bouzy. Associating domain-dependent knowledge and Monte Carlo approaches within a go program. *Information Sciences*, 175(4):247–257, 2005.
4. D.M. Breuker, Uiterwijk J.W.H, H, and H.J. van den Herik. Replacement Schemes and Two-Level Tables. *ICCA Journal*, 19(3):175–180, 1996.
5. T. Cazenave. Multi-player Go. In H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands, editors, *Computers and Games (CG 2008)*, volume 5131 of *LNCS*, pages 50–59, Berlin, Germany, 2008. Springer.
6. G.M.J-B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, and B. Bouzy. Progressive strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, 4(3):343–357, 2008.

7. R. Coulom. Efficient selectivity and backup operators in Monte-Carlo Tree Search. In H.J. van den Herik, P. Ciancarini, and H.H.L.M. Donkers, editors, *Computers and Games (CG 2006)*, volume 4630 of *LNCS*, pages 72–83, Berlin, Germany, 2007. Springer.
8. H. Finnsson and Y. Björnsson. Simulation Control in General Game Playing Agents. In *IJCAI'09 Workshop on General Intelligence in Game Playing Agents*, pages 21–26, 2009.
9. D.E. Knuth and R.W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.
10. L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Artificial Intelligence (LNAI)*, pages 282–293, Berlin, Germany, 2006. Springer.
11. R.J. Lorentz. Improving Monte-Carlo Tree Search in Havannah. In H.J. van den Herik, H. Iida, and A. Plaat, editors, *Computers and Games (CG 2010)*, volume 6515 of *LNCS*, pages 105–115, Berlin, Germany, 2011. Springer.
12. U. Lorenz and T. Tscheuschner. Player Modeling, Search Algorithms and Strategies in Multi-player Games. In H.J. van den Herik, S.-C. Hsu, T.-S. Hsu, and H.H.L.M. Donkers, editors, *Advances in Computer Games (ACG11)*, volume 4250 of *LNCS*, pages 210–224, Berlin, Germany, 2006. Springer.
13. C. Luckhart and K.B. Irani. An algorithmic solution of n-person games. In *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 158–162, 1986.
14. T.A. Marsland. A review of game-tree pruning. *ICCA Journal*, 9(1):3–19, 1986.
15. J.A.M. Nijssen and M.H.M. Winands. Enhancements for Multi-Player Monte-Carlo Tree Search. In H.J. van den Herik, H. Iida, and A. Plaat, editors, *Computers and Games (CG 2010)*, volume 6515 of *LNCS*, pages 238–249, Berlin, Germany, 2011. Springer.
16. A. Rimmel, F. Teytaud, and O. Teytaud. Biasing Monte-Carlo Simulations through RAVE Values. In H.J. van den Herik, H. Iida, and A. Plaat, editors, *Computers and Games (CG 2010)*, volume 6515 of *LNCS*, pages 59–68, Berlin, Germany, 2011. Springer.
17. S. Sackson. *A Gamut of Games*. Random House, New York, NY, USA, 1969.
18. M.P.D. Schadd and M.H.M. Winands. Best Reply Search for Multiplayer Games. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):57–66, 2011.
19. N.R. Sturtevant. An analysis of UCT in multi-player games. In H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands, editors, *Computers and Games (CG 2008)*, volume 5131 of *LNCS*, pages 37–49, Berlin, Germany, 2008. Springer.
20. N.R. Sturtevant and R.E. Korf. On pruning techniques for multi-player games. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 201–207. AAAI Press / The MIT Press, 2000.
21. M.H.M. Winands and Y. Björnsson. $\alpha\beta$-based Play-outs in Monte-Carlo Tree Search. In *2011 IEEE Conference on Computational Intelligence and Games (CIG 2011)*, pages 110–117. IEEE Press, 2011.
22. M.H.M. Winands, Y. Björnsson, and J.-T. Saito. Monte Carlo Tree Search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):239–250, 2010.