Enhancements for Multi-Player Monte-Carlo Tree Search

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) is becoming increasingly popular for playing multi-player games. In this paper we propose two enhancements for MCTS in multi-player games: (1) Progressive History and (2) Multi-Player Monte-Carlo Tree Search Solver (MP-MCTS-Solver). We analyze the performance of these enhancements in two different multi-player games: Focus and Chinese Checkers. Based on the experimental results we conclude that Progressive History is a considerable improvement in both games and MP-MCTS-Solver, using the standard update rule, is a genuine improvement in Focus.

1 Introduction

Multi-player board games are games that can be played by more than two players. In the past the standard techniques to play these games were $\max^n$ [11] and Paranoid [16]. For computers, even with these techniques, multi-player games are generally more difficult than two-player games [14]. There are two main reasons for this. The first reason is that pruning in game trees is more difficult. With $\alpha\beta$ pruning, the size of a tree in a two-player game can be reduced from $O(b^d)$ to $O(b^{\frac{d}{2}})$ in the best case. However, when using $\max^n$, safe pruning is hardly possible. In Paranoid, the size of the game tree can only be reduced to $O(b^{\frac{n-1}{n}d})$ in the best case. If the number of players is large enough, there will be hardly any reduction. The second reason is coalition forming. Contrary to two-player games, where two players always play against each other, in multi-player games coalitions might occur. This can change the behavior of the opponents, making it more difficult to predict their preferred moves during search.

Over the past years, Monte-Carlo Tree Search (MCTS) [6, 9] has become increasingly popular for letting computers play board games. It has been applied successfully in 2-player games such as Go [5–7], Amazons [8, 10], Lines of Action [18] and Hex [4]. Sturtevant [15] showed that MCTS outperforms $\max^n$ and Paranoid in the multi-player game Chinese Checkers. Moreover, Cazenave [3] applied MCTS successfully for multi-player Go.

In this paper we propose two new enhancements for MCTS in multi-player games. The first one is Progressive History, a combination of Progressive Bias

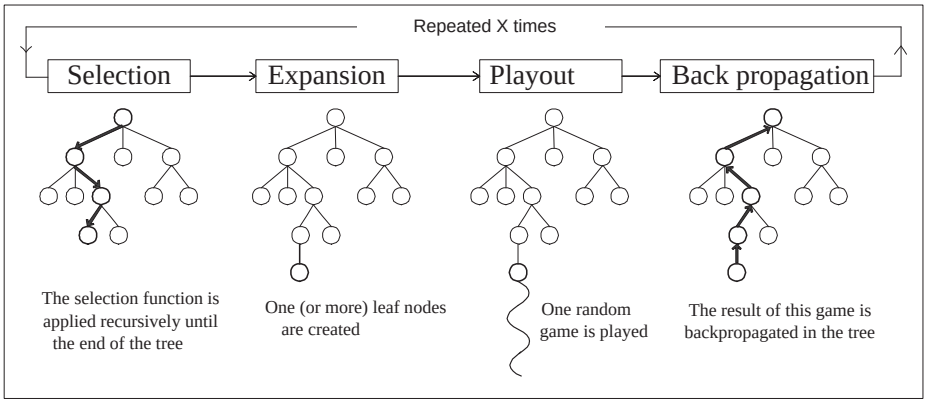


Fig. 1. Monte-Carlo Tree Search scheme [5].

[5] and the history heuristic [13]. The second one is a multi-player variant of Monte-Carlo Tree Search Solver [19], called Multi-Player Monte-Carlo Tree Search Solver (MP-MCTS-Solver). We test these enhancements in two different multi-player games: Focus and Chinese Checkers.

The remainder of this paper is structured as follows. In Sect. 2 we describe MCTS and the two enhancements for multi-player games. In Sect. 3, a brief explanation of the games that we use as test domain is given. The experiments and the results are given in Sect. 4. Finally, in Sect. 5 we give the conclusions that can be drawn from our research and we give an outlook on future research.

2 Monte-Carlo Tree Search Enhancements

In this section we give a brief overview of MCTS and two enhancements. In Subsection 2.1 we briefly discuss MCTS. Next, we propose Progressive History in Subsection 2.2. Finally, in Subsection 2.3 MP-MCTS-Solver is introduced.

2.1 MCTS

MCTS [6, 9] is a best-first search technique that uses Monte-Carlo simulations to guide the search. MCTS consists of four phases (see Fig. 1). We explain them in detail below.

Selection. The first phase is the selection phase. Here, 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) [9] selection strategy. The child i with the highest score v_i is selected as follows (Formula 1)

$$v_i = \frac{s_i}{n_i} + C \times \sqrt{\frac{\ln(n_p)}{n_i}}, \quad (1)$$

here 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.

Expansion. The second phase is the expansion phase. Here, 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 [6].

Playout. The third phase is the playout phase. During this phase, moves are played in self-play until the game is finished. Usually, the playouts are being generated using random move selection. However, it is also possible to add domain knowledge to the playouts. Sturtevant [15] proposed to use a strategy, called the ϵ-greedy strategy [17], in which the algorithm chooses the most greedy move (the best move according to a simple move evaluation function) with a probability of $1 - \epsilon$. A random move is selected with a probability of ϵ. In our program we use $\epsilon = 0.05$.

Backpropagation. 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 of the winning player is 1, the value of the other players is 0. MCTS is able to handle games with multiple winners. For instance, if Player 1 and Player 2 both win in a 3-player game, then the tuple $[\frac{1}{2}, \frac{1}{2}, 0]$ is returned. The multi-player games we use in this paper, Chinese Checkers and Focus, cannot have multiple winners.

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

2.2 Progressive History

A problem with MCTS is that it takes a while before enough information is gathered to calculate a somewhat reliable value for a node. Chaslot *et al.* [5] proposed Progressive Bias to direct the search according to – possibly time-expensive – heuristic knowledge. They added to Formula 1 the following component: $\frac{H_i}{n_i+1}$. Here H_i represents heuristic knowledge, which depends only on the board configuration represented by the node i. The influence of this component is important when a few number of games has been played, but decreases fast (when more

games have been played) to ensure that the strategy converges to a pure selection strategy such as UCT.

The problem of Progressive Bias is that heuristic knowledge is needed. A solution is offered by using the (relative) history heuristic [13, 20], which is used in MCTS enhancements such as RAVE [7] and Gibbs sampling [1]. The history heuristic does not require any domain knowledge. The idea behind the history heuristic is to exploit the fact that moves that are good in a position are also good in other positions. For each move that has been performed for each player during the simulations, the number of games and the total score are stored. This information is used to compute the history score. This score is subsequently combined – in a more complex way than Progressive Bias – with the UCT selection strategy.

In this paper we propose a new enhancement, called Progressive History, that combines Progressive Bias and the history heuristic. The heuristic knowledge H_i of Progressive Bias is replaced with the history score. The child i with the highest score v_i is selected now as follows (Formula 2)

$$v_i = \frac{s_i}{n_i} + C \times \sqrt{\frac{\ln(n_p)}{n_i}} + \frac{s_a}{n_a} \times \frac{W}{n_i - s_i + 1}, \tag{2}$$

here s_a represents the score of move a, where each game in which s_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 game in the past. W is a constant which determines the influence of Progressive History. The higher the value of W, the longer Progressive History affects the selection of the node.

In Formula 2, $\frac{W}{n_i - s_i + 1}$ represents the Progressive Bias part and $\frac{s_a}{n_a}$ the history heuristic part. We remark that, in the Progressive Bias part, we do not divide by the number of visits as standard is done [5, 19], but by the number of visits minus the score, i.e. the number of losses. In this way, nodes that do not perform well are not biased too long, whereas nodes that continue to have a high score stay biased. To ensure that we do not divide by 0, a 1 is added in the denominator.

In our implementation the move data for Progressive History is stored in a global table, while RAVE [7] has to keep track of the "all-move-as-first" (AMAF) [2] values in every node. Keeping track of the values globally instead of locally at every node saves memory space, but has the risk that it diminishes the benefit. Another solution to save space is to define move categories, e.g. capture moves. The disadvantage of this solution is that it is dependent on domain knowledge.

2.3 Multi-Player MCTS-Solver

Recently, Winands *et al.* [19] developed a method, called Monte-Carlo Tree Search Solver (MCTS-Solver), to prove the game-theoretical value of a node in a Monte-Carlo search tree. This method was used successfully for playing the two-player game Lines of Action [12].

We developed a multi-player variant of MCTS-Solver, called Multi-Player Monte-Carlo Tree Search Solver (MP-MCTS-Solver). For the multi-player vari-

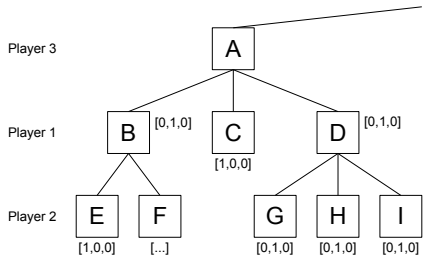


Fig. 2. A multi-player search tree

ant, 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 at one of the children a win is found 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 labeled as a win for this opponent. However, if the children lead to wins for different opponents, then updating the game-theoretical values becomes a non-trivial task. Update rules have to be developed to take care of such situations. We tested 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 simulation score is used.

An example is given in Fig. 2. Here, node E is a terminal node where Player 1 has won. It means that node B is a mate-in-1 for Player 1, regardless of the value of node F. This node receives a game-theoretical value of $[1, 0, 0]$.[1] Nodes G, H, and I all result in wins for Player 2. Then parent node D receives a game-theoretical value of $[0, 1, 0]$, since this node always leads to a win for the same opponent of Player 1. The game-theoretical value of node A cannot be determined in this case, because both Player 1 and Player 2 can win and there is no win for Player 3.

(2) The *paranoid* update rule uses the assumption that the opponents of the root player will never let him win [3, 16]. Again consider Fig. 2. Assuming that the root player is Player 1, using the paranoid update rule, we can determine the game-theoretical value of node A. Since we assume that Player 3 will not let Player 1 win, the game-theoretical value of node A becomes $[0, 1, 0]$.

Note that if there are still multiple winners after removing the root player from the list of possible winners, then no game-theoretical value is assigned to the node and the simulation score is used.

[1] If a node has a game-theoretical value, then this value is used in the selection phase, without using either UCT or Progressive History.

The paranoid update rule may not always give the desired result. With the paranoid assumption, the game-theoretical 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, since the game-theoretical 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-theoretical value of $[0, 1, 0]$. This is actually false, 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-theoretical value which is technically false. 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 will not get the win and, as a result, will not overestimate the position. When using the first-winner update rule, in Fig. 2 node A will receive the game-theoretical value $[1, 0, 0]$.

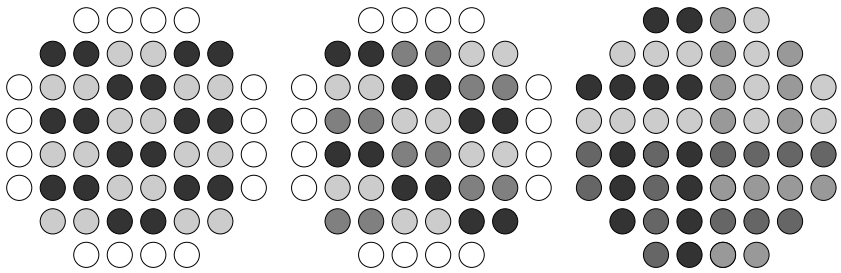
3 Test Domains

We tested the two enhancements 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 3.1 and 3.2, respectively.

3.1 Focus

Focus is an abstract multi-player strategy board game, invented in 1963 by Sid Sackson [12]. This game has also been released under the name *Domination*.

Focus is played on an 8×8 board where in each corner 3 squares 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. 3, the initial board positions for the 2-, 3- and 4-player variants are given.



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

Fig. 3. Set-ups for Focus

Each turn a player may move a stack, which contains one or more pieces, orthogonally as many squares 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. Players are also allowed to split stacks in two smaller stacks. If they decide to do so, then they only move the upper stack as many squares as the number of pieces that are being moved.

If a stack lands on 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 of his pieces back on the board, instead of moving a 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, games can take a long time to finish. Therefore, we decided to use the shortened version of the game. In this version, a player has won if he has either captured a total number of pieces, 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. Finally, in the 4-player variant, the goal is to capture at least 2 pieces from all opponents or capture at least 10 pieces in total.

3.2 Chinese Checkers

Chinese Checkers is a board game that can be played by 2 to 6 players. This game has been invented in 1893 and has since then been released by various publishers 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 [14] (see Fig. 4). 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 us to use a stronger evaluation function (see Subsection 4.1).

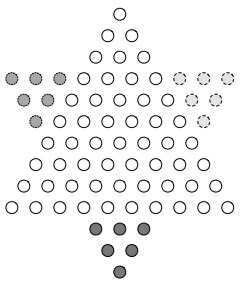


Fig. 4. A Chinese Checkers board [14].

The goal of each player is to move all his pieces to his own base at the other side of the board. Pieces may move to one of the adjacent squares or they may jump over another piece to an empty square. It is also allowed to make multiple jumps with one piece in one turn. It is 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 Experiments

In this section, we describe the experiments that were performed to investigate the strength of the proposed enhancements to MCTS in Focus and Chinese Checkers. In Subsection 4.1 we explain the experimental setup. In Subsection 4.2 we describe the experiments and give the results of Progressive History. In Subsection 4.3, the experiments and the results for MP-MCTS-Solver are given.

4.1 Experimental Setup

The Monte-Carlo engines of Focus and Chinese Checkers have been written in Java. MCTS, Progressive History, and MP-MCTS-Solver all set their exploration constant C to 0.2. For the playouts, we apply the ϵ-greedy strategy [15], with a 95% probability to play the greediest move and a 5% probability to play a random move. For determining the greediest move in Focus, we apply a simple evaluation function which assigns a value to each valid move. This value is based on the number of captured pieces and the amount of stacks that the players control after moving. For determining the most greedy move in Chinese Checkers, we use a lookup table [15]. In this table we store, for each possible configuration of pieces, the minimum number of moves a player should perform before he can have all pieces in his home base, assuming that there are no opponents' pieces on the board. During the mid-game this value is not very accurate, though it is still useful. In the end-game, however, it leads to optimal play.

In all experiments, the players received 2.5 seconds thinking time to determine the best move. All experiments were performed on an AMD64 2.4 GHz. computer. For reference, in Focus around 4,000 games per second are played at the start of the game. During the end-game around 20,000 games per second are played. In Chinese Checkers, the number of games per second is slightly lower. At the start of the game around 3,000 games per second are played. During the end-game, up to 10,000 games per second are played.

For both games, there may be an advantage regarding the order of play and the number of different players (i.e. search configurations). To give reliable results, each possible player setup, with the exception of setups where each player is of the same search configuration, is played equally often.

Table 1. Win rates for a Progressive History player with different values of W against the default MCTS player in Focus.

	2 players			3 players			4 players		
W	wins	losses	win rate	wins	losses	win rate	wins	losses	win rate
0	1746	1614	52.0%	1720	1640	51.2%	1706	1654	50.8%
0.05	1983	1367	59.0%	2054	1306	61.1%	1931	1429	57.5%
0.1	2009	1351	59.8%	2116	1244	63.0%	1978	1382	58.9%
0.25	2061	1299	61.3%	2115	1245	62.9%	1997	1363	59.4%
0.5	2154	1206	64.1%	2200	1160	65.5%	2013	1347	59.9%
1	2219	1141	66.0%	2196	1164	65.4%	1957	1403	58.2%
3	2089	1271	62.2%	2190	1170	65.2%	2002	1358	59.6%
5	1946	1414	57.9%	2143	1217	63.8%	2001	1359	59.6%
7.5	1722	1638	51.3%	2036	1324	60.6%	1917	1443	57.1%
10	1593	1767	47.4%	1941	1419	57.8%	1911	1449	56.9%

Table 2. Win rates for a Progressive History player with different values of W against the default MCTS player in Chinese Checkers.

	2 players			3 players			4 players		
W	wins	losses	win rate	wins	losses	win rate	wins	losses	win rate
0.25	1773	1587	52.8%	1981	1379	59.0%	1913	1447	56.9%
0.5	1955	1405	58.2%	2110	1250	62.8%	1960	1400	58.3%
1	2279	1081	67.8%	2132	1228	63.5%	2079	1281	61.9%
3	2683	677	79.9%	2242	1118	66.7%	2232	1128	66.4%
5	2804	556	83.5%	2211	1149	65.8%	2244	1116	66.8%
10	2795	565	83.2%	2193	1167	65.3%	2337	1023	69.6%
15	2721	639	81.0%	2183	1177	65.0%	2326	1034	69.2%
20	2044	1316	60.8%	2022	1338	60.2%	2124	1236	63.2%

4.2 Progressive History

In the following series of experiments we tested Progressive History (with different values of W) against an MCTS player without Progressive History in Focus and Chinese Checkers.

Table 1 shows that Progressive History, provided the value of W is set correctly, is a considerable improvement for MCTS in Focus. The best result for the 2-player variant is achieved with W=1, achieving a win rate of 66.0%. For the 3-player variant, the best results are achieved with W=1 and W=3, both winning more than 65% versus an MCTS player without Progressive History. In the 4-player variant, Progressive History still performs well. With W=3 or W=5 the win rate is still almost 60%.

Table 2 reveals that Progressive History works even better in Chinese Checkers. In the 2-player variant, Progressive History easily wins over 80% of the games, with the best result achieved by the player with W=5, winning 83.5% of the games. In the 3-player game, the win rate drops to around 65%, but in the 4-player game, the performance increases again to almost 70% with W=10 or W=15.

Table 3. Win rates for a Progressive History player using $\frac{W}{n_i - s_i + 1}$ against a Progressive History player using $\frac{W}{n_i + 1}$.

	2 players			3 players			4 players		
Game	wins	losses	win rate	wins	losses	win rate	wins	losses	win rate
Focus	680	370	64.8%	640	410	61.0%	546	504	52.0%
Chinese Checkers	582	468	57.6%	575	475	54.8%	566	484	53.9%

The reason why Progressive History works so well in Chinese Checkers is that for this game good moves are not dependent on the global board state. Good moves are often moves that move a checker far forward, and thus are good moves in similar, but different states. In Focus, this is much less the case. Good moves are considerably more dependent on the global board state. Still, Progressive History is an important enhancement in Focus as well.

In the next series of experiments we verified whether dividing by the number of losses ($\frac{W}{n_i - s_i + 1}$) instead of the number of games ($\frac{W}{n_i + 1}$) in the Progressive Bias part of Formula 2 is an improvement. In Table 3 the results are given when the two players played against each other in Focus and Chinese Checkers. The players used W=5, which is in both cases one of the best settings. We see that for both games dividing by the number of losses is an improvement for Progressive History. However, the performance drops when the number of players increases.

4.3 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. We performed these experiments in Focus, because MCTS-Solver is only successful in sudden-death games [19]. Chinese Checkers is not a sudden-death game, and therefore we expect MP-MCTS-Solver not to work well in this game. However, Focus is a sudden-death game and is therefore an appropriate test domain for MP-MCTS-Solver.

In the last series of experiments, Progressive History was enabled for both players with W=5. In Table 4, 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 well. For the 2-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

Table 4. Win rates for an MP-MCTS-Solver player with different update rules against the default MCTS player in Focus.

	2 players			3 players			4 players		
Type	wins	losses	win rate	wins	losses	win rate	wins	losses	win rate
Standard	1780	1580	53.0%	1844	1516	54.9%	1792	1568	53.3%
Paranoid	1745	1615	51.9%	1693	1667	50.4%	1510	1850	44.9%
First-winner	1774	1586	52.8%	1732	1628	51.5%	1457	1903	43.4%

overhead. In the 3-player variant, the performance drops to just over 50% for both. In the 4-player variant, the win rate of the player using MP-MCTS-Solver is even 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.

5 Conclusions and Future Research

In this paper we investigated two enhancements for MCTS in multi-player games. The first one is Progressive History, a combination of Progressive Bias and the history heuristic. The second one is MP-MCTS-Solver, a multi-player variant of MCTS-Solver. We determined the strength of these enhancements in two different games: Focus and Chinese Checkers.

For Progressive History, we determined its strength by letting it play with different values of the constant W against an MCTS player without Progressive History. Depending on the game, the number of players and the value of W, Progressive History wins 60% to 70% of the games against MCTS without Progressive History. Based on these results, we may conclude that Progressive History is an important enhancement for MCTS in multi-player games.

We tested MP-MCTS-Solver with three different update rules, namely (1) standard, (2) paranoid and (3) first-winner. We tested this enhancement only in Focus, since MP-MCTS-Solver only works well in sudden-death games. Chinese Checkers is, contrary to Focus, not a sudden-death game. A win rate between 53% and 55% was achieved in Focus with the standard update rule. The other two update rules achieved similar win rates in the 2-player variant, but were around or below 50% for the 3- and 4-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 multi-player games, there is still much room for improvement. Progressive History works well in Focus and Chinese Checkers and may also work well in other games. This is subject of future research. Moreover, comparisons with other variants to bias the selection strategy, such as RAVE [7], Gibbs sampling [1] and prior knowledge [7] should be performed. 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. For instance, one could try to create new update rules that may improve its performance.

References

1. 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.
2. B. Brügmann. Monte Carlo Go. Technical report, Physics Department, Syracuse University, 1993. `ftp://ftp.cse.cuhk.edu.hk/pub/neuro/GO/mcgo.tex`.
3. 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 *Lecture Notes in Computer Science (LNCS)*, pages 50–59, Berlin, Germany, 2008. Springer.

4. T. Cazenave and A. Saffidine. Utilisation de la recherche arborescente Monte-Carlo au Hex. *Revue d'Intelligence Artificielle*, 23(2–3):183–202, 2009. In French.
5. 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.
6. 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 *Lecture Notes in Computer Science (LNCS)*, pages 72–83, Berlin, Germany, 2007. Springer.
7. S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM.
8. J. Kloetzer, H. Iida, and B. Bouzy. Playing amazons endgames. *ICGA Journal*, 32(3):140–148, 2009.
9. 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*, pages 282–293, 2006.
10. R.J. Lorentz. Amazons discover Monte-Carlo. In H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands, editors, *Computers and Games (CG 2008)*, volume 5131 of *Lecture Notes in Computer Science (LNCS)*, pages 13–24, Berlin, Germany, 2008. Springer.
11. 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.
12. S. Sackson. *A Gamut of Games*. Random House, New York, NY, USA, 1969.
13. J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.
14. 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 *Lecture Notes in Computer Science (LNCS)*, pages 37–49, Berlin, Germany, 2008. Springer.
15. N.R. Sturtevant. An analysis of UCT in multi-player games. *ICGA Journal*, 31(4):195–208, 2008.
16. 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.
17. R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.
18. M.H.M. Winands and Y. Björnsson. Evaluation function based Monte-Carlo LOA. In H.J. van den Herik and P.H.M. Spronck, editors, *Advances in Computer Games (ACG 2009)*, volume 6048 of *Lecture Notes in Computer Science (LNCS)*, pages 33–44, Berlin, Germany, 2010. Springer.
19. M.H.M. Winands, Y. Björnsson, and J-T. Saito. Monte-Carlo Tree Search Solver. In H.J. van den Herik, X. Xu, Z. Ma, and M.H.M. Winands, editors, *Computers and Games (CG 2008)*, volume 5131 of *Lecture Notes in Computer Science (LNCS)*, pages 25–36, Berlin, Germany, 2008. Springer.
20. M.H.M. Winands, E.C.D. van der Werf, H.J. van den Herik, and J.W.H.M. Uiterwijk. The relative history heuristic. In H.J. van den Herik, Y. Björnsson, and N.S. Netanyahu, editors, *Computers and Games*, volume 3846 of *Lecture Notes in Computer Science (LNCS)*, pages 262–272, Berlin, Germany, 2006. Springer.