Enhancements for Multi-Player Monte-Carlo Tree Search[1]

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

Games and AI Group, Department of Knowledge Engineering
Maastricht University, Maastricht, The Netherlands

In this abstract we propose two new enhancements for Monte-Carlo Tree Search (MCTS) in multi-player games. The first one is Progressive History, a combination of Progressive Bias [2] and the history heuristic [4]. The second one is a multi-player variant of Monte-Carlo Tree Search Solver [6], 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.

A problem of MCTS is that it takes a while before enough information is gathered to calculate a reliable value for a node. Chaslot *et al.* [2] proposed Progressive Bias (a UCT variant [3]) to direct the search according to heuristic knowledge. We propose an enhancement, called Progressive History, that combines Progressive Bias and the history heuristic. The child i with the highest score v_i is selected as follows

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

where s_i denotes the total score of child i, and s_a represents the score of the move a corresponding to node i. The variables n_i and n_p denote the total number of times that child i and parent p have been visited, respectively. n_a is the number of times move a has been played in any game in the past. C is a constant that determines the exploration factor of UCT [3] and W is a constant that determines the influence of Progressive History. In Formula 1, $\frac{W}{n_i - s_i + 1}$ 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 [2, 6], 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 the first 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. Table 2 reveals that Progressive History works even better in Chinese Checkers. Moreover, additional experiments revealed that dividing by the number of losses increased the performance of Progressive History.

Table 1: Progressive History player with different W values vs. default MCTS player in Focus.

	2 players			3 players			4 players		
W	wins	losses	win rate	wins	losses	win rate	wins	losses	win rate
0.1	2009	1351	59.8%	2116	1244	63.0%	1978	1382	58.9%
1	2219	1141	66.0%	2196	1164	65.4%	1957	1403	58.2%
5	1946	1414	57.9%	2143	1217	63.8%	2001	1359	59.6%
10	1593	1767	47.4%	1941	1419	57.8%	1911	1449	56.9%

Recently, Winands *et al.* [6] 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 in the two-player game Lines of Action. We developed a multi-player variant of MCTS-Solver,

[1]The full version of this paper will be published in: *Computers and Games (CG 2010)*, Lecture Notes in Computer Science, Springer, Heidelberg, Germany.

Table 2: Progressive History player with different W values vs. 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
1	2279	1081	67.8%	2132	1228	63.5%	2079	1281	61.9%
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%
20	2044	1316	60.8%	2022	1338	60.2%	2124	1236	63.2%

called Multi-Player Monte-Carlo Tree Search 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. 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: (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. (2) The *paranoid* update rule uses the assumption that the opponents of the root player will never let him win [1, 5]. 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. Problems may 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 gives the win to the player who is the first winner after him. In this way the player before him does not receive the win and, as a result, does not overestimate the position.

In the second series of experiments, we tested MP-MCTS-Solver with the three different update rules playing against the default MCTS player. We performed these experiments in Focus, because MCTS-Solver is only successful in sudden-death games [6]. 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. Progressive History was enabled for both players with W=5. In Table 3, we see that in Focus the standard update rule works best.

Table 3: MP-MCTS-Solver player with different update rules vs. 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%

Based on the results, we may conclude that Progressive History is an important enhancement for MCTS in multi-player games. MP-MCTS-Solver with the standard update rule performs well. The other two update rules, paranoid and first-winner, were not successful in Focus.

References

[1] 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.

[2] 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.

[3] 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 *LNAI*, pages 282–293, 2006.

[4] J. Schaeffer. The history heuristic. *ICCA Journal*, 6(3):16–19, 1983.

[5] N.R. Sturtevant and R.E. Korf. On pruning techniques for multi-player games.

[6] 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 *LNCS*, pages 25–36, Berlin, Germany, 2008. Springer.