Generalized Rapid Action Value Estimation

Tristan Cazenave

Generalized Rapid Action Value Estimation

Tristan Cazenave
LAMSADE - Universite Paris-Dauphine
Paris, France
cazenave@lamsade.dauphine.fr

Abstract

Monte Carlo Tree Search (MCTS) is the state of the art algorithm for many games including the game of Go and General Game Playing (GGP). The standard algorithm for MCTS is Upper Confidence bounds applied to Trees (UCT). For games such as Go a big improvement over UCT is the Rapid Action Value Estimation (RAVE) heuristic. We propose to generalize the RAVE heuristic so as to have more accurate estimates near the leaves. We test the resulting algorithm named GRAVE for Atarigo, Knighthrough, Domineering and Go.

1 Introduction

Monte Carlo Tree Search (MCTS) is a general search algorithm that was initially designed for the game of Go [Coulom, 2006]. The most popular MCTS algorithm is Upper Confidence bounds applied to Trees (UCT) [Kocsis and Szepesvári, 2006]. All modern computer Go programs use MCTS. It has increased the level of Go programs up to the level of the strongest amateur players. Rapid Action Value Estimation (RAVE) [Gelly and Silver, 2007; 2011] is commonly used in Go programs as it is a simple and powerful heuristic.

MCTS and RAVE are also used in MoHex the best Hex playing program [Arneson *et al.*, 2010]. Adding the RAVE heuristic to MoHex produces a 181 Elo strength gain.

Another successful application of MCTS is General Game Playing (GGP). The goal of GGP is to play games unknown in advance and to design algorithms able to play well at any game. An international GGP competition is organized every year at AAAI [Genesereth *et al.*, 2005]. The best GGP programs use MCTS [Finnsson and Björnsson, 2008; Méhat and Cazenave, 2011]. In this paper we propose an algorithm that can be applied to many games without domain specific knowledge. Hence it is of interest to GGP engines.

MCTS can also be applied to other problems than games [Browne *et al.*, 2012]. Examples of non-games applications are Security, Mixed Integer Programming, Traveling Salesman Problem, Physics Simulations, Function Approximation, Constraint Problems, Mathematical Expressions, Planning and Scheduling.

The paper is organized in three remaining sections: section two presents related works for games, MCTS and RAVE, section three details the GRAVE algorithm and section four gives experimental results for various numbers of playouts and various sizes of Atarigo, Knightthrough, Domineering and Go.

2 Related Work

UCT is the standard MCTS algorithm. It uses the mean of the previous random playouts to guide the beginning of the current playouts. There is a balance between exploration and exploitation when choosing the next move to try at the beginning of a playout. Exploitation tends to choose the move with the best mean, while exploration tends to try alternative and less explored moves to see if they can become better. The principle of UCT is optimism in face of uncertainty. It chooses the move with the UCB formula, m is a possible move:

$$argmax_m(mean_m + c \times \sqrt{\frac{log(playouts)}{playouts_m}})$$

The c exploration parameter has to be tuned for each problem. Low values encourage exploitation while high values encourage exploration.

The All Moves As First heuristic (AMAF) [Bouzy and Helmstetter, 2003] is a heuristic that was used in Gobble, the first Monte Carlo Go program [Brügmann, 1993]. It consists in updating the statistics of the moves of a position with the result of a playout, taking into account all the moves that were played in the playout and not only the first one.

RAVE [Gelly and Silver, 2007; 2011] is an improvement of UCT that was originally designed for the game of Go and that works for multiple games. It consists in memorizing in every node of the tree the statistics for all possible moves even if they are not yet tried in the node. When a playout is over, the statistics of all the nodes it has traversed are updated with all the moves of the playout. Another way to describe it is that the AMAF value of each possible move is recorded in a node. When there are few playouts, the AMAF value is used to choose a move. When the number of playouts increases the weight of the mean increases and the weight of AMAF decreases. The formula to choose the move can be stated with a weight β_m, p_m is the number of playouts starting with move m and $pAMAF_m$ is the number of playouts containing

move m:

$$\beta_m \leftarrow \frac{pAMAF_m}{pAMAF_m + p_m + bias \times pAMAF_m \times p_m}$$

$$argmax_m((1.0 - \beta_m) \times mean_m + \beta_m \times AMAF_m)$$

In their paper [Gelly and Silver, 2011] Sylvain Gelly and David Silver state: "In principle it is also possible to incorporate the AMAF values, from ancestor subtrees. However, in our experiments, combining ancestor AMAF values did not appear to confer any advantage.". On the contrary, we found that it can be useful.

RAVE is used in many computer Go programs. Examples of such programs are Mogo [Lee *et al.*, 2009] and Fuego [Enzenberger *et al.*, 2010].

RAVE has also been used for GGP in CadiaPlayer [Finnsson and Björnsson, 2010]. CadiaPlayer with RAVE is much stronger at Breakthrough, Checkers and Othello than the usual MCTS. On the contrary it is slightly weaker with RAVE at Skirmish.

An alternative to RAVE for Go programs is to use learned patterns and progressive widening [Coulom, 2007; Ikeda and Viennot, 2014]. The principle is to learn a weight for each pattern. This weight can be used in playouts to bias the policy and in the tree to select promising moves. Progressive widening starts with only the best ranked move as a possible move in a node and then increases the number of moves that can be tried as the number of playouts of the node also increases. It is related to GRAVE since it only tries the most promising moves when there are few playouts.

In Mogo, an improvement of RAVE is to start with heuristic values for moves sampled only a few times [Lee *et al.*, 2009]. The heuristic value is computed after patterns that match around the move. The score of a move is calculated according to three weights, α, β and γ. The weight for the mean of the playouts is α, it starts at zero for few playouts and then increases as more playouts are played. The weight for the AMAF mean is β. It starts at zero then increases to 1 with more playouts and then decreases to zero with even more playouts. The last weight is γ, it starts with a value greater than 1 and then decreases as more playouts occur.

The GRAVE algorithm also uses an estimate different from the node's AMAF mean for moves that have only few playouts. But on the contrary of the γ weight associated to a valued pattern it does not use domain specific and learned knowledge. It is more simple and more general.

Another improvement of Mogo is to use the RAVE values of moves during the playouts [Rimmel *et al.*, 2011]. This improvement gives good results for the game of Havannah and for the game of Go. The algorithm descends the tree using the usual RAVE algorithm and when it is outside the tree it uses the RAVE values of the last node with at least 50 simulations. During the playout it chooses one of the k moves with the best RAVE values with a given probability. Otherwise it uses the usual playout policy.

3 GRAVE

RAVE computes an estimate of each possible move when only a few playouts have been played. The principle of GRAVE is to use AMAF values of a state higher up in the tree than the current state. There is a balance between the accuracy of an estimate and the accordance to the current state of the estimate. A state upper in the tree has better accuracy since it has more associated playouts. However the statistics are about a different state some moves before and are less to the point for the lower state in the tree than the statistics of lower states. GRAVE principle is to only use statistics that were calculated with more than a given number of playouts. We take as a reference state the closest ancestor state that has more playouts than a given ref constant. The reference state can be the current state if the number of playouts of the current state is greater than ref.

The GRAVE algorithm is given in algorithm 1. The algorithm is close to the RAVE algorithm. The main difference is that it uses a $tref$ parameter that contains the transposition table entry of the last node with more than ref playouts. The ref constant is to be tuned as well as the bias. This $tref$ entry is used to get the RAVE statistics instead of the usual t entry. If the number of playouts of a node is greater than ref then $tref$ is equal to t and the algorithm behaves as usual RAVE. If the number of playouts of the current node is lower than ref then the algorithm uses the last entry along the path to the node that has a number of playouts greater than ref. This entry is named $tref$ and is used to compute the AMAF values at the current node.

Instead of only updating the AMAF statistics for the color and the moves of the node, GRAVE updates statistics for all possible moves by both colors at every node.

GRAVE is a generalization of RAVE since GRAVE with ref equals to zero is RAVE.

We will now detail algorithm 1. It starts with calculating the possible moves that will be used later to find the most promising move. If the board is terminal, the game is over and the score is returned. The algorithm does not explicitly represent the tree. Instead it uses a transposition table to remember all the visited states and stores in each entry of the transposition table the number of wins of each move, the number of playouts of each move, the number of AMAF wins for all possible moves in the entire game and not only in the state, that is including the opponent player moves, the number of AMAF playouts for all possible moves in the entire game. These values stored in the variables w, p, wa, pa are used to calculate the β_m parameter as well as the $AMAF$ and $mean$ values of a move. The usual RAVE formula is then used to calculate the value of a move. The move with the best value is played and the algorithm is recursively called. The recursion ends when reaching a state that is not in the transposition table. In this case the state is added in the transposition table, a playout is played and the transposition table entry is updated with the result of the playout.

The only difference to the usual RAVE algorithm is the $tref$ parameter and the ref constant. Instead of using the usual t entry of the transposition table in order to calculate the AMAF value of a move, the algorithm calculates the AMAF

value using the $tref$ entry. The $tref$ entry is updated using the condition $t.playouts > ref$. It means that $tref$ contains the last entry in the recursive calls with a number of playouts greater than ref.

Algorithm 1 The GRAVE algorithm

```
GRAVE (board, tref)
moves ← possible moves
if board is terminal then
  return score(board)
end if
t ← entry of board in the transposition table
if t exists then
  if t.playouts > ref then
    tref ← t
  end if
  bestValue ← −∞
  for m in moves do
    w ← t.wins[m]
    p ← t.playouts[m]
    wa ← tref.winsAMAF[m]
    pa ← tref.playoutsAMAF[m]
    β_m ← pa / (pa+p+bias×pa×p)
    AMAF ← wa/pa
    mean ← w/p
    value ← (1.0 − β_m) × mean + β_m × AMAF
    if value > bestValue then
      bestValue ← value
      bestMove ← m
    end if
  end for
  play(board, bestMove)
  res ← GRAVE(board, tref)
  update t with res
else
  t ← new entry of board in the transposition table
  res ← playout(player, board)
  update t with res
end if
return res
```

4 Experimental Results

In order to test GRAVE we have used the following method: the RAVE bias is first tuned playing different RAVE bias against UCT with a 0.4 exploration parameter (the exploration parameter used for GGP), then the GRAVE bias as well as the ref constant are tuned against the tuned RAVE. The resulting GRAVE algorithm is then tested against RAVE with different bias to make sure there is no over-fitting nor a miss of a good RAVE bias and that GRAVE performs well against all possible RAVE. In order to tune both RAVE and GRAVE we test as bias all the powers of 10 between 10^{-1} and 10^{-15}. Additionally the ref constants tested for GRAVE are 25, 50, 100, 200 and 400.

The tested games are Atarigo 8×8, Atarigo 19×19, Knightthrough 8×8, Domineering 8×8, Domineering 19×19, Go 9×9, Go 19×19 and three color Go. The algorithms are tested for 1,000 and 10,000 playouts. For two-player games each result is the average winning rate over 500 games, 250 playing first and 250 playing second.

Table 1: Tuning the RAVE bias against UCT at Atarigo 8×8 with 10,000 playouts.

bias	0.1	0.01	0.001	10^{-4}	10^{-5}
	74.4 %	91.2 %	90.6 %	92.8 %	91.4 %
bias	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
	92.8 %	**94.2 %**	91.4 %	91.6 %	91.0 %
bias	10^{-11}	10^{-12}	10^{-13}	10^{-14}	10^{-15}
	93.8 %	92.2 %	92.2 %	92.2 %	92.2 %

Table 2: GRAVE against RAVE at Atarigo 8×8 with 10,000 playouts.

ref	25	50	100
bias	10^{-8}	10^{-10}	10^{-3}
	86.2 %	**88.4 %**	87.8 %

4.1 Atarigo 8×8

Atarigo is a simplification of the game of Go. The winner of a game is the first player to capture a string of stones.

Atarigo has been solved up to size 6×6 with threat based algorithms [Cazenave, 2003; Boissac and Cazenave, 2006]. In this paper we use the 8×8 board size.

All algorithms use 10,000 playouts. We first tuned the RAVE bias against UCT. Results are given in table 1. The best bias is 10^{-7} and it wins 94.2 % of the games against UCT.

We then tuned GRAVE against the best RAVE. Results are given in table 2. The best GRAVE has a ref equals to 50 and a bias of 10^{-10}, it wins 88.4 % of the games against RAVE.

When playing GRAVE with ref equals to 50 and bias equals to 10^{-10} against all RAVE bias, the worst score obtained by GRAVE was 85.2 % against a RAVE bias of 10^{-4}.

4.2 Atarigo 19×19

We played RAVE with 1,000 playouts and with various bias against UCT with 1,000 playouts. The results are in table 3. The best result is 72.4 % for a bias of 0.001.

We then played GRAVE with different bias and ref against the best RAVE. The results are in table 4. The best GRAVE wins 78.2 % with ref equals to fifty and a 10^{-5} bias.

Table 3: Tuning the RAVE bias against UCT at Atarigo 19×19 with 1,000 playouts.

bias	0.1	0.01	0.001	10^{-4}	10^{-5}
	63.6 %	67.6 %	**72.4** %	73.2 %	68.6 %
bias	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
	67.8 %	66.8 %	67.0 %	67.8 %	70.4 %
bias	10^{-11}	10^{-12}	10^{-13}	10^{-14}	10^{-15}
	70.4 %	70.4 %	70.4 %	70.4 %	70.4 %

Table 4: GRAVE against RAVE at Atarigo 19×19 with 1,000 playouts.

ref	50	100	200
bias	10^{-5}	10^{-5}	10^{-8}
	78.2 %	74.4 %	75.2 %

Table 5: Tuning the RAVE bias against UCT at Knightthrough 8×8 with 1,000 playouts.

bias	0.1	0.01	0.001	10^{-4}	10^{-5}
	63.4 %	**69.4 %**	65.2 %	69.4 %	68.2 %
bias	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
	68.8 %	69.0 %	68.6 %	69.4 %	65.0 %
bias	10^{-11}	10^{-12}	10^{-13}	10^{-14}	10^{-15}
	64.4 %	66.0 %	66.0 %	66.0 %	66.0 %

4.3 Knightthrough 8×8

Knightthrough is played on a 8×8 chess board. The initial position has two rows of white knights on the bottom and two rows of black knights on the top. Players alternate moving a knight as in chess except that the knight can only go forward and not backward. Captures can occur as in chess. The first player to move a knight on the last row of the opposite side has won. Knightthrough is a game from the GGP competitions, it is derived from Breakthrough that is a similar game with pawns.

Table 5 gives the results of RAVE against UCT. Each player is allocated 1,000 playouts at each move. The best bias is 0.01 as it wins 69.4 % against UCT.

Table 6 gives the results for GRAVE with different ref values against RAVE with a bias of 0.01. They both use 1,000 playouts for each move. GRAVE with ref equals to 50 and a bias of 10^{-4} wins 67.8 % against RAVE.

Table 7 gives the results of RAVE against UCT for different bias and 10,000 playouts each. The best bias is 10^{-10} as it wins 56.2 % against UCT.

Table 8 gives the results for GRAVE with different ref values against RAVE with a bias of 10^{-10}. Each player uses 10,000 playouts per move. The best GRAVE player with ref equals to 50 and a bias of 0.001 wins 67.2 % against the tuned RAVE.

4.4 Domineering 8×8

Domineering is a two player combinatorial game usually played on an 8×8 board. It consists in playing 2×1 dominoes on the board. The first player put the dominoes vertically and the second player put them horizontally. If a player cannot play anymore, he loses the game.

Domineering was invented by Göran Andersson [Gardner, 1974]. As it decomposes in independent parts it was studied

Table 6: GRAVE against RAVE at Knightthrough 8×8 with 1,000 playouts.

ref	25	50	100
bias	0.01	10^{-4}	0.01
	62.0 %	**67.8 %**	63.0 %

Table 7: Tuning the RAVE bias against UCT at Knightthrough 8×8 with 10,000 playouts.

bias	0.1	0.01	0.001	10^{-4}	10^{-5}
	38.0 %	55.2 %	52.6 %	53.8 %	53.8 %
bias	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
	49.4 %	52.8 %	53.6 %	53.6 %	**56.2 %**
bias	10^{-11}	10^{-12}	10^{-13}	10^{-14}	10^{-15}
	52.0 %	52.0 %	52.0 %	52.0 %	52.0 %

Table 8: GRAVE against RAVE at Knightthrough 8×8 with 10,000 playouts.

ref	25	50	100
bias	10^{-9}	0.001	0.001
	65.8 %	**67.2 %**	65.4 %

by the combinatorial games community. They solved it for small boards [Lachmann *et al.*, 2002]. Boards up to 10×10 were solved using $\alpha\beta$ search [Bullock, 2002]. Recently a knowledge based method was proposed that can solve large rectangular boards without any search [Uiterwijk, 2014].

In order to tune the RAVE bias we played RAVE against UCT with 10,000 playouts. The best RAVE bias is 10^{-3} which wins 72.6 % of the time against UCT as can be seen in table 9.

We then played different GRAVE algorithms against the tuned RAVE. The results are given in table 10. With ref equals to 25 and a bias of 10^{-5}, GRAVE wins 62.4 % of the time against the tuned RAVE.

When playing GRAVE with ref equals to 25 and bias equals to 10^{-5} against all RAVE bias, the worst score obtained by GRAVE was 58.2 % against a RAVE bias of 10^{-4}.

4.5 Domineering 19×19

In order to tune the RAVE bias for Domineering 19×19 we ran the experiment RAVE with 1,000 playouts against UCT with 1,000 playouts, but RAVE won all of its games with all bias. So we ran another experiment: RAVE with 1,000 playouts against UCT with 10,000 playouts. The results are given in table 11. The best score is 63.8 % obtained for a bias of 10^{-7}.

We tuned GRAVE against the best RAVE algorithm. Results are given in table 12. The best score is 56.4 % with ref equals to 100 and a bias of 10^{-6}.

Table 9: Tuning the RAVE bias against UCT at Domineering 8×8 with 10,000 playouts.

bias	0.1	0.01	0.001	10^{-4}	10^{-5}
	50.6 %	71.2 %	**72.6 %**	68.4 %	68.0 %
bias	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
	67.8 %	71.6 %	70.6 %	69.2 %	65.2 %
bias	10^{-11}	10^{-12}	10^{-13}	10^{-14}	10^{-15}
	69.2 %	69.2 %	69.2 %	69.2 %	69.2 %

Table 10: GRAVE against RAVE at Domineering 8×8 with 10,000 playouts.

ref	25	50	100
bias	10^{-5}	10^{-9}	10^{-10}
	62.4 %	60.4 %	59.2 %

Table 11: Tuning the RAVE bias with 1,000 playouts against UCT with 10,000 playouts at Domineering 19×19.

bias	0.1	0.01	0.001	10^{-4}	10^{-5}
	28.0 %	51.4 %	59.6 %	59.0%	60.8%
bias	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
	60.2 %	**63.8 %**	59.6 %	62.8 %	60.0 %
bias	10^{-11}	10^{-12}	10^{-13}	10^{-14}	10^{-15}
	62.4 %	62.4 %	62.4 %	62.4 %	62.4 %

4.6 Go 9×9

Go is an ancient oriental game of strategy that originated in China thousands of years ago [Bouzy and Cazenave, 2001; Müller, 2002]. It is usually played on a 19×19 grid. Players alternate playing black and white stones on the intersections of the grid. The goal of the game using Chinese rules is to have more stones on the board than the opponent at the end of the game. There is a capture rule: when a string of stones of the same color is surrounded by stones of the opposing color, the surrounded string is removed from the board. There are a lot of Go players in China, Japan and Korea and hundreds of professional players. Go is the last of the popular board games that is better played by humans than by computers. The best computer Go players are currently four stones behind the best Go players.

MCTS has been very successful for the game of Go. The original RAVE algorithm was designed for computer Go [Gelly and Silver, 2007].

This section deals with Go 9×9, the next section is about Go 19×19.

The playout policy we used in our experiments for Go 9×9 and Go 19×19 is Mogo style playouts with patterns [Gelly *et al.*, 2006].

We have first tuned RAVE with 1,000 playouts against UCT with 1,000 playouts. The results are given in table 13. The best bias is 10^{-7} and it wins 89.6 % of the games against UCT.

We have then played GRAVE against the tuned RAVE. The results are given in table 14. The best GRAVE configuration is ref equals to 100 and a bias of 10^{-6}. It wins 66.0 % against the tuned RAVE.

We repeated the experiments for 10,000 playouts. The RAVE bias was tuned against UCT. The results are in table 15. The best bias for RAVE is 10^{-4} and it wins 73.2 % against UCT. GRAVE was then played against the tuned RAVE. The results are given in table 16. GRAVE with a bias of 10^{-4} and ref equals to 50 wins 54.4 % against RAVE.

Table 12: GRAVE against RAVE at Domineering 19×19 with 1,000 playouts.

ref	50	100	200	400
bias	10^{-7}	10^{-6}	10^{-9}	10^{-8}
	55.6 %	**56.4 %**	56.0 %	53.8 %

Table 13: Tuning the RAVE bias against UCT at Go 9×9 with 1,000 playouts.

bias	0.1	0.01	0.001	10^{-4}	10^{-5}
	83.6 %	88.2 %	87.0 %	89.6 %	87.2 %
bias	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
	87.0 %	**89.6 %**	88.0 %	86.2 %	86.2 %
bias	10^{-11}	10^{-12}	10^{-13}	10^{-14}	10^{-15}
	86.2 %	86.2 %	86.2 %	86.2 %	86.2 %

Table 14: GRAVE against RAVE at Go 9×9 with 1,000 playouts.

ref	25	50	100	200
bias	10^{-8}	10^{-10}	10^{-6}	10^{-6}
	65.2 %	62.6 %	**66.0 %**	61.2 %

4.7 Go 19×19

When tuning the RAVE bias against UCT with 1,000 playouts, all results were greater than 98 %. We therefore tuned RAVE with 1,000 playouts against GRAVE with 1,000 playouts, ref equals to 400 and a $bias$ of 10^{-5}. The worst result for GRAVE was 72.2 % against a RAVE bias of 10^{-5}. We then tuned GRAVE against RAVE with a bias of 10^{-5}. The results are given in table 17. The best result is 81.8 % with ref equals to 100 and a bias of 10^{-9}.

The best RAVE bias for 1,000 playouts was reused to tune GRAVE with 10,000 playouts against RAVE with 10,000 playouts. We only tuned GRAVE with ref equals to 100. The best result for GRAVE was 73.2 % with a bias of 10^{-12}. We then played GRAVE with a bias of 10^{-12} and ref equals to 100 against all RAVE with bias between 10^{-1} and 10^{-12}. The worst result for GRAVE was 62.4 % against the 10^{-7} bias.

4.8 Three Color Go 9×9

Multicolor Go is a variation of the game of Go which is played with more than two players. For example it is possible to add a third color, say red, and to play with stones of three different colors [Cazenave, 2008]. The rules are the same as in the Chinese version of the game. At the end of the game when all three players have passed, the score for a given color is the number of stones of the color on the board

Table 15: Tuning the RAVE bias against UCT at Go 9×9 with 10,000 playouts.

bias	0.1	0.01	0.001	10^{-4}	10^{-5}
	29.0 %	69.4 %	**73.2 %**	62.8 %	68.6 %
bias	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
	64.6 %	64.6 %	63.2 %	63.8 %	67.0 %
bias	10^{-11}	10^{-12}	10^{-13}	10^{-14}	10^{-15}
	63.6 %	68.4 %	68.4 %	68.4 %	68.4 %

Table 16: GRAVE against RAVE at Go 9×9 with 10,000 playouts.

ref	25	50	100	200
bias	10^{-5}	10^{-4}	10^{-5}	10^{-5}
	52.4 %	**54.4 %**	51.6 %	45.4 %

Table 17: GRAVE against RAVE at Go 19×19 with 1,000 playouts.

ref	50	100	200	400
bias	10^{-7}	10^{-9}	10^{-11}	10^{-6}
	78.2 %	**81.8 %**	80.0 %	77.6 %

plus the number of eyes of the color. The winner is the player that has the greatest score at the end.

In order to test an algorithm we make it play the six possible combinations of colors against two other algorithms, one hundred times. This results in six hundreds games played. The percentage of wins for algorithms that are close in strength is therefore close to 33.3 %. An algorithm scoring 50.0 % is already much better than the two other algorithms.

The motivation for testing GRAVE at Three Color Go is that playouts contain less moves of a given color than in usual two player Go. The AMAF statistics take more playouts to be accurate than in two player Go. So GRAVE which uses more accurate statistics may perform better.

Mogo style playouts do not work well for multicolor Go, so for these experiments we use a uniform random playout policy without patterns.

We tuned the RAVE bias with 1,000 playouts against two UCT players each with 1,000 playouts. The results are given in table 18. The best RAVE bias is 0.001 with 70.83 % wins. This is a clear win for RAVE since it is well above the standard score of 33.33 %.

We played GRAVE against two tuned RAVE with a bias of 0.001. The results are given in Table 19. The best GRAVE has a ref equals to 100 and a bias of 10^{-5} and it wins 57.17 % against a tuned RAVE.

4.9 Three Color Go 19×19

For Three Color Go on a 19×19 board, RAVE is much better than UCT when they both use 1,000 playouts. We therefore did the same as in Go 19×19 and tuned RAVE with 1,000 playouts against two players using GRAVE with 1,000 playouts, ref equals to 400 and a $bias$ of 10^{-5}. The best result for RAVE was 18.5 % with a bias of 10^{-9}.

Table 18: Tuning the RAVE bias against UCT at Three Color Go 9×9 with 1,000 playouts.

bias	0.1	0.01	0.001	10^{-4}	10^{-5}
	49.7 %	70.5 %	**70.8 %**	66.7 %	68.2 %
bias	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
	67.3 %	64.5 %	67.8 %	63.5 %	64.7 %
bias	10^{-11}	10^{-12}	10^{-13}	10^{-14}	10^{-15}
	63.3 %	63.3 %	63.3 %	63.3 %	63.3 %

Table 19: GRAVE against RAVE at Three Color Go 9×9 with 1,000 playouts.

ref	50	100	200	400
bias	10^{-4}	10^{-5}	10^{-9}	10^{-11}
	56.00 %	**57.17 %**	54.50 %	55.17 %

5 Conclusion

We have presented a generalization of RAVE named GRAVE. It uses the AMAF values of an ancestor node when the number of playouts is too low to have meaningful AMAF statistics. It uses a threshold on the number of playouts of the node to decide whether to use the current node's statistics or the ancestor node's statistics. It is a generalization of RAVE since GRAVE with a threshold of zero is RAVE.

GRAVE is better than RAVE and UCT for Atarigo, Knightthrough, Domineering and Go.

For Atarigo 8×8 the results show that GRAVE is a large improvement over RAVE since GRAVE wins 85.2 % against RAVE when they both use 10,000 playouts. For Atarigo 19×19 and 1,000 playouts GRAVE wins 78.2 %

For Knightthrough 8×8 GRAVE with 1,000 playouts wins 67.8 % against RAVE with 1,000 playouts. GRAVE with 10,000 playouts wins 67.2 % against RAVE with 10,000 playouts. RAVE is a large improvement over UCT for Knightthrough 8×8 and GRAVE is a large improvement over RAVE.

For Domineering 8×8 GRAVE with 10,000 playouts wins 62.4 % against RAVE with 10,000 playouts. For Domineering 19×19 GRAVE with 1,000 playouts wins 56.4 % against RAVE with 1,000 playouts.

For Go 9×9 GRAVE with 1,000 playouts wins 66.0 % against RAVE with 1,000 playouts. For 10,000 playouts it wins 54.4 %. For Go 19×19 GRAVE with 10,000 playouts wins 62.4 % against RAVE with 10,000 playouts.

For Three Color Go 9×9, GRAVE wins 57.17 % against a tuned RAVE. For Three Color Go 19×19, RAVE only wins 18.5 % against an unoptimized GRAVE.

GRAVE is a simple and generic improvement over RAVE. It works for at least four games without game specific knowledge.

Acknowledgments

This work was granted access to the HPC resources of MesoPSL financed by the Region Ile de France and the project Equip@Meso (reference ANR-10-EQPX-29-01) of the programme Investissements d'Avenir supervised by the Agence Nationale pour la Recherche

References

[Arneson *et al.*, 2010] Broderick Arneson, Ryan B Hayward, and Philip Henderson. Monte carlo tree search in hex. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):251–258, 2010.

[Boissac and Cazenave, 2006] Frédéric Boissac and Tristan Cazenave. De nouvelles heuristiques de recherche appliquées à la résolution d'atarigo. In *Intelligence artificielle et jeux*, pages 127–141. Hermes Science, 2006.

[Bouzy and Cazenave, 2001] Bruno Bouzy and Tristan Cazenave. Computer Go: An AI oriented survey. *Artif. Intell.*, 132(1):39–103, 2001.

[Bouzy and Helmstetter, 2003] Bruno Bouzy and Bernard Helmstetter. Monte-Carlo Go developments. In *ACG*, volume 263 of *IFIP*, pages 159–174. Kluwer, 2003.

[Browne *et al.*, 2012] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, March 2012.

[Brügmann, 1993] Bernd Brügmann. Monte Carlo Go. Technical report, 1993.

[Bullock, 2002] Nathan Bullock. Domineering: Solving large combinatorial search spaces. *ICGA Journal*, 25(2):67–84, 2002.

[Cazenave, 2003] Tristan Cazenave. A generalized threats search algorithm. In *Computers and Games*, volume 2883 of *Lecture Notes in Computer Science*, pages 75–87. Springer, 2003.

[Cazenave, 2008] Tristan Cazenave. Multi-player go. In *Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 - October 1, 2008. Proceedings*, volume 5131 of *Lecture Notes in Computer Science*, pages 50–59. Springer, 2008.

[Coulom, 2006] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, pages 72–83, 2006.

[Coulom, 2007] Rémi Coulom. Computing elo ratings of move patterns in the game of go. *ICGA Journal*, 30(4):198–208, 2007.

[Enzenberger *et al.*, 2010] Markus Enzenberger, Martin Müller, Broderick Arneson, and Richard Segal. Fuego - an open-source framework for board games and Go engine based on Monte Carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):259–270, 2010.

[Finnsson and Björnsson, 2008] Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *AAAI*, pages 259–264, 2008.

[Finnsson and Björnsson, 2010] Hilmar Finnsson and Yngvi Björnsson. Learning simulation control in general game-playing agents. In *AAAI*, pages 954–959, 2010.

[Gardner, 1974] Martin Gardner. Mathematical games. *Scientific American*, 230:106–108, 1974.

[Gelly and Silver, 2007] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*, pages 273–280, 2007.

[Gelly and Silver, 2011] Sylvain Gelly and David Silver. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artif. Intell.*, 175(11):1856–1875, 2011.

[Gelly *et al.*, 2006] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in monte-carlo go. 2006.

[Genesereth *et al.*, 2005] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the aaai competition. *AI magazine*, 26(2):62, 2005.

[Ikeda and Viennot, 2014] Kokolo Ikeda and Simon Viennot. Efficiency of static knowledge bias in Monte-Carlo tree search. In *Computers and Games*, pages 26–38. Springer, 2014.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *17th European Conference on Machine Learning (ECML'06)*, volume 4212 of *LNCS*, pages 282–293. Springer, 2006.

[Lachmann *et al.*, 2002] Michael Lachmann, Cristopher Moore, and Ivan Rapaport. Who wins domineering on rectangular boards. *More Games of No Chance*, 42:307–315, 2002.

[Lee *et al.*, 2009] Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, J-B Hoock, Arpad Rimmel, F Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, and Tzung-Pei Hong. The computational intelligence of MoGo revealed in Taiwan's computer Go tournaments. *Computational Intelligence and AI in Games, IEEE Transactions on*, 1(1):73–89, 2009.

[Méhat and Cazenave, 2011] Jean Méhat and Tristan Cazenave. A parallel general game player. *KI-Künstliche Intelligenz*, 25(1):43–47, 2011.

[Müller, 2002] Martin Müller. Computer go. *Artif. Intell.*, 134(1-2):145–179, 2002.

[Rimmel *et al.*, 2011] Arpad Rimmel, Fabien Teytaud, and Olivier Teytaud. Biasing monte-carlo simulations through rave values. In *Computers and Games*, pages 59–68. Springer, 2011.

[Uiterwijk, 2014] Jos WHM Uiterwijk. Perfectly solving domineering boards. In *Computer Games*, pages 97–121. Springer, 2014.