On the tactical and strategic behaviour of MCTS when biasing random simulations

Fabien Teytaud, Julien Dehos

On the tactical and strategic behaviour of MCTS when biasing random simulations

Fabien Teytaud* Julien Dehos†

Abstract

Over the last few years, many new algorithms have been proposed to solve combinatorial problems. In this field, Monte-Carlo Tree Search (MCTS) is a generic method which performs really well on several applications; for instance, it has been used with notable results in the game of Go. To find the most promising decision, MCTS builds a search tree where the new nodes are selected by sampling the search space randomly (i.e., by Monte-Carlo simulations). However, efficient Monte-Carlo policies are generally difficult to learn. Even if an improved Monte-Carlo policy performs adequately in some games, it can become useless or harmful in other games depending on how the algorithm takes into account the *tactical* and the *strategic* elements of the game.

In this article, we address this problem by studying *when* and *why* a learned Monte-Carlo policy works. To this end, we use (1) two known Monte-Carlo policy improvements (PoolRave and Last-Good-Reply) and (2) two connection games (Hex and Havannah). We aim to understand how the benefit is related (a) to the number of random simulations and (b) to the various game rules (within them, tactical and strategic elements of the game). Our results indicate that improved Monte-Carlo policies, such as PoolRave or Last-Good-Reply, work better for games with a strong tactical element for small numbers of random simulations, whereas more general policies seem to be more suited for games with a strong strategic element for higher numbers of random simulations.

1 Introduction

Monte-Carlo Tree Search (MCTS) algorithms have had a huge impact on Artificial Intelligence [KS06, Cou07, CSB+06]. They are now generally used for decision-making problems such as games [GS07, Lor08, Caz09, AHH10, TT09, Stu15, HS14] and planning problems [KS06, NM09]. One of the most interesting advantages of the MCTS algorithms is generality. Indeed, since they use Monte-Carlo simulations, they do not require an evaluation function (see Section 3). This makes them well-suited to applications for which expert knowledge is difficult to obtain [Fin12, WM14]. A second advantage is that MCTS algorithms are anytime, meaning they can be stopped at any moment and return the best decision found so far [KS06]. However, they can be costly for some applications. Furthermore, increasing the number of simulations (i.e., giving more time to compute the search) does not guarantee to obtain better results [BCC+10].

The principle of MCTS is to build a subtree of possible future states of a problem, and to perform many random simulations, called "playouts", in order to evaluate these states. Numerous improvements have been proposed. Well-known examples involve biasing the decisions made to traverse the subtree; for instance, by using the Rapid Action Value Estimate (RAVE) [GS07] or adding expert knowledge [CFH+10, CHTT09]. Improvements on the Monte-Carlo policy (used for performing playouts) are often more complicated. Indeed, a great deal of studies have been performed with a general success, in trying to learn such a policy. However, the difficulty is that improving the Monte-Carlo policy does not guarantee that the general behaviour of the resulting MCTS will be better.

*LISIC, ULCO, Université Lille Nord-de-France, France, email:teytaud@lisic.univ-littoral.fr
†LISIC, ULCO, Université Lille Nord-de-France, France

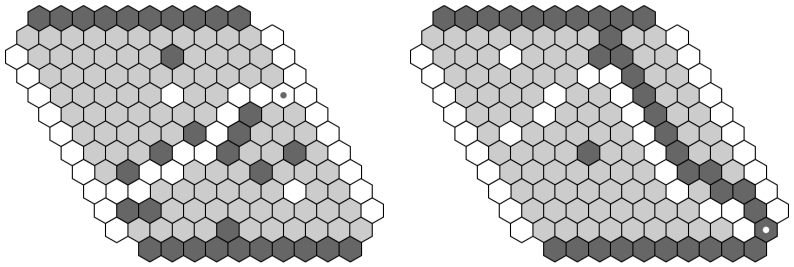

Figure 1: Winning Hex: white player wins (left); black player wins (right).

In this article, we aim at understanding *why* and *when* a random simulation biasing technique works. To this end, we study two known Monte-Carlo policy improvements (PoolRave and Last-Good-Reply) using two games (Hex and Havannah). Whereas these games have many similarities (they both belong to the family of connection games), they are still differently suited to the Monte-Carlo policy improvements. Therefore, we perform a large number of experiments and analyze a variety of algorithmic parameters and game rules to explain the gain (or loss) that the improvements of the Monte-Carlo policy can bring to MCTS.

The article is organised as follows. Section 2 presents the two application games used in this study (Hex and Havannah). Section 3 presents the MCTS algorithm and three of its well-known improvements (RAVE, PoolRave, Last-Good-Reply). Section 4 contains experiments and results of MCTS on the two application games. In Section 5, we modify one of the applications in order to study the tactical and strategic behaviour of the Monte-Carlo learnings. Section 6 provides our conclusions and points to a future research direction.

2 Games of Hex and Havannah

Below we briefly describe the two application games used (Sections 2.1 and 2.2). In Section 2.3 we distinguish the tactical and strategic elements in these games.

2.1 Hex

The game of Hex is a 2-player board game. It has been created independently by Piet Hein and John Nash during the 1940s. It is played on a hexagonal grid, traditionally on a 11x11 or 14x14 rhombus. Both players put alternately a stone in an empty location on the board. Players must form a chain of stones of their colour between the two board sides of their colour (see Fig. 1). It is proved that there is no possible draw, thus one player must win [Ber76, Gal79]. Moreover, for this game we know that the first player must have a winning strategy. A way to prove this is to use a strategy stealing argument: the game is finite with perfect information, and draws are not possible, so one of the two players must have a winning strategy. It is important to add that having a stone on the board cannot be a drawback. Then, if the second player has a winning strategy, the first player can play a random move and steal the strategy of the second player. This ensures a first player win.

Many studies have been performed using artificial intelligence for the game of Hex. Currently, the best algorithms for this game combine a Monte-Carlo Tree Search algorithm and a solver [AHH10, AHH09]. Because Hex is played and studied by many people, expert knowledge exists and can been use: for instance, with patterns or opening books [Maa05]. All this is beyond the scope of this article. Here, we are only interested in the Monte-Carlo Tree Search itself, and in particular in the possible learning of the Monte-Carlo part.

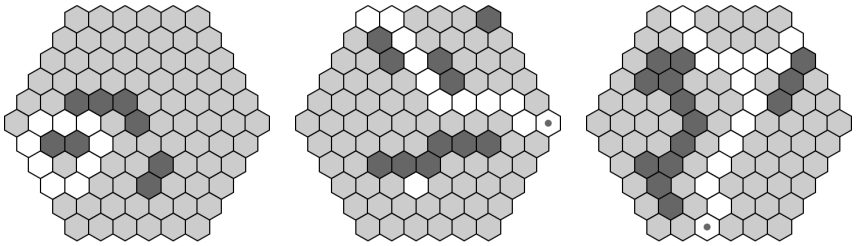

Figure 2: The three winning shapes of Havannah (for the white player): ring (left), bridge (middle) and fork (right).

2.2 Havannah

The game of Havannah is a 2-player board game created by Christian Freeling [Sch92]. It is played on an hexagonal board of hexagonal cells. There are 6 corners and 6 edges, but an important point is that corner stones do not belong to edges. Both players play alternately a stone in an empty location on the board. If there is no more empty location and if no player wins then it is a draw. In order to win, a player has to realise one of these three shapes (see Fig. 2).

- A ring, which is a loop around one or more cells (empty or not, occupied by black or white stones).
- A bridge, which is a continuous string of stones connecting two corners.
- A fork, which is a continuous string of stones connecting three edges (edges exclude corners).

Havannah is known to be a difficult game for artificial intelligence, especially because there is no intuitive evaluation function and because of the large state space. Previous studies on this game include using Monte-Carlo Tree Search algorithms [TT09, Lor10, Ewa12].

2.3 Tactical and strategic elements in games

As far as we know, the notions of tactic and strategy first appeared in a military context. In this context, a strategy corresponds to the definition of a global plan (or objectives) that will lead to the victory. A tactic corresponds to the different decisions and threats used to gain this plan (or these objectives).

The notions of tactic and strategy also appear in games and have a similar definition. They are notably important for a human player to improve his or her playing. In games, we often talk about long-term plan (strategy) and short-term decisions (tactics). For instance, in Chess, a strategic plan could be to have a better pawn structure in the middle-game or in the end-game. Tactics are short-term decisions in order to achieve this previous plan (for instance, by threatening a piece of higher value).

In this paper, we focus on these notions in the game of Havannah: expert players state that rings correspond to tactical aspects (it is really rare to win with a ring, but rings can be used as local threats) whereas forks are generally considered as the strategic aspect (experts tend to build a global winning plan thanks to a fork).

3 Monte-Carlo Tree Search based methods

In this section we give a general outline of MCTS in 3.1. Then in 3.2 to 3.4 we briefly discuss three policy improvements, viz. Rapid Action Value Estimate (RAVE), PoolRave, and Last-Good-Reply.

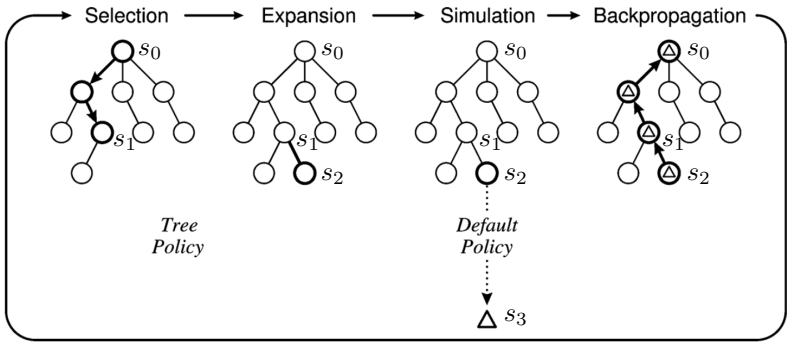

Figure 3: The MCTS algorithm iteratively builds a subtree of the possible next states (circles) of the current problem. This figure illustrates one iteration of the algorithm. Starting from the root node s_0 (current state of the problem), a node s_1 is selected and a new node s_2 is created. A random simulation is computed (until a final state s_3 is reached) and the subtree is updated.

3.1 Monte-Carlo Tree Search

Below, we present the Monte-Carlo Tree Search algorithm (MCTS) and some of its enhancements. MCTS is currently a state-of-the-art algorithm for decision making problems [KS06, Cou07, CSB+06]. It is particularly relevant in games [GS07, Lor08, Caz09, AHH10, TT09].

From the current state s_0 of a problem, the MCTS algorithm computes the best possible following state to choose. To estimate this best choice, the algorithm iteratively constructs a subtree of possible futures of the problem (i.e., successive possible states, starting from s_0), using random simulations. Each iteration of the construction of the subtree is done in four steps, namely: *selection*, *expansion*, *simulation* and *backpropagation* (see Fig. 3 from [BPW+12]). The four steps are given in pseudo code in Algorithm 1.

The *selection* step consists in choosing the first node found with untried child actions in the subtree. In the most common implementation of MCTS, called Upper Confidence Tree (UCT), the subtree is traversed, from the root node to a leaf node, using a bandit formula:

$$s_1 \leftarrow \underset{j \in C_{s_1}}{\arg\max} \left[\bar{r}_j + C \sqrt{\frac{\ln(n_{s_1})}{n_j}} \right],$$

where C_{s_1} is the set of child nodes of the node s_1, $\bar{r}_j$ is the average reward for the node j (the ratio of the number of wins over the number of simulations), n_j is the number of simulations for the node j, and n_{s_1} is the number of simulations for the node s_1 ($n_{s_1} = \sum_j n_j$). C is called the exploration parameter and is used to tune the trade-off between exploitation and exploration.

Once a leaf node s_1 is selected (i.e., a node of which all possible decisions have not been considered, in child nodes), the next step (*expansion*) consists in creating a new child node s_2 corresponding to a possible decision of s_1 which has not been considered yet.

The next step (*simulation*) consists in performing a random game (i.e., each player plays randomly until a final state s_3 is reached), leading to a reward r (win, loss or possibly draw).

Finally (*backpropagation*), r is used to update the reward in the newly created node and in all the nodes which have been encountered during the *selection* (see Algorithm 1):

$$\bar{r}_{s_4} \leftarrow \frac{\bar{r}_{s_4} \times (n_{s_4} - 1) + r}{n_{s_4}},$$

where s_4 is the node to update, n_{s_4} is the number of simulations for s_4 and $\bar{r}_{s_4}$ is the average reward for s_4.

Algorithm 1 : Monte-Carlo Tree Search

{initialization}
$s_0 \leftarrow$ create root node from the current state of the problem

while there is some time left **do**

{selection}
$s_1 \leftarrow s_0$
while all possible decisions of s_1 have been considered **do**
$C_{s_1} \leftarrow$ child nodes of s_1
$s_1 \leftarrow \underset{j \in C_{s_1}}{\arg\max} \left[\bar{r}_j + C\sqrt{\frac{\ln(n_{s_1})}{n_j}} \right]$
{expansion}
$s_2 \leftarrow$ create a child node of s_1 from a possible decision of s_1 not yet considered

{simulation}
$s_3 \leftarrow s_2$
while s_3 is not a terminal state for the problem **do**
$s_3 \leftarrow$ randomly choose next state from s_3

{backpropagation}
$r \leftarrow$ reward of the terminal state s_3
$s_4 \leftarrow s_2$
while $s_4 \neq s_0$ **do**
$n_{s_4} \leftarrow n_{s_4} + 1$
$\bar{r}_{s_4} \leftarrow \frac{\bar{r}_{s_4} \times (n_{s_4} - 1) + r}{n_{s_4}}$
$s_4 \leftarrow$ parent node of s_4

return best child of s_0

3.2 Rapid Action Value Estimate

The Rapid Action Value Estimate (RAVE) improvement [GS07] is based on the idea of permutation of moves. If a move is good in a certain situation, then it may be good in another one. Thus, to make a decision in the subtree, a RAVE score is added to the bandit formula defined in Section 3.1. For this improvement, it is necessary to keep more information for each node in the subtree. Let us define $m = f \rightarrow s$, meaning the move m played in situation f leads to situation s. For each node it is necessary to store the following four numbers.

- The number of wins by playing m in f (already needed for the bandit formula).
- The number of losses by playing m in f (already needed for the bandit formula).
- The number of wins when m is played after the situation f (but not necessarily in f), called RAVE wins.
- The number of losses when m is played after the situation f (but not necessarily in f), called RAVE losses.

The numbers of RAVE wins/losses for a move m are large compared to their classic numbers of wins/losses. The variance for estimating the move m will then be lower, but the bias will be higher (as this is an approximation: the move is not played in f). The idea is to use especially the RAVE score when the number of simulations is small. Its influence progressively decreases as the number of simulations increases.

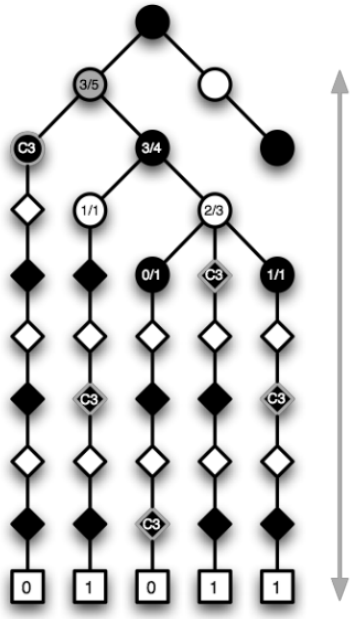

Figure 4: The RAVE improvement consists in biasing the selection step of the MCTS algorithm by adding a RAVE score in the bandit formula. The RAVE score of a move is the win ratio of the move played in other simulations of the subtree. For example, selecting the move c_3 using RAVE (top left circle), is biased with the win ratio of c_3 in the other branches of the subtree (diamonds).

In order to select a move in the subtree, the bandit formula is replaced by:

$$s_1 \leftarrow \underset{j \in C_{s_1}}{\arg\max} \left[(1-\beta)\ \bar{r}_j + \beta\ \bar{r}_{s_1,j}^{RAVE} + C\sqrt{\frac{\ln(n_{s_1})}{n_j}} \right],$$

where β is a parameter which tends to 0 as the number of simulations increases. According to [GS07], we can use $\beta = \sqrt{\frac{R}{R+3n_j}}$ where R is a parameter to tune (see Fig. 4).

3.3 PoolRave

The PoolRave enhancement [RTT10] is based on the RAVE improvement. The idea is the same as the RAVE idea, except that instead of using the RAVE values to bias the *selection* step (tree policy), it is used to bias the *simulation* step (Monte-Carlo policy). It can easily be combined with the RAVE improvement.

To compute a simulation step for a node n, with the PoolRave improvement, a pool of possible moves is built first, using the RAVE scores of n: for each possible move m of the node n, m is added to the pool if its RAVE score is meaningful (i.e., the number of RAVE simulations of m is greater than a threshold); then the pool is sorted according to the RAVE score and only the N best moves of the pool are kept. Thus, the pool contains the N moves of n which seem to be the best moves according to RAVE (i.e., moves that have the best ratios of RAVE wins over RAVE losses). Once the pool is built, the simulation step consists in playing moves chosen preferably in the pool: a move is chosen in the pool with a probability p, otherwise (with probability $1-p$) a random move is played as in the classic MCTS algorithm. The size of the pool (N) and the probability of playing a PoolRave move (p) are parameters to be tuned.

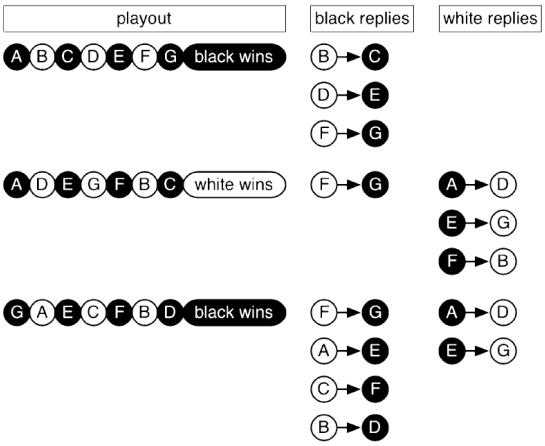

Figure 5: The LGRF-1 improvement (simulation step biasing). For black replies: during a first playout, replies C to the move B, E to the move D and G to the move F are learned. During the next simulation, the replies C to B and E to D are played. These two replies are deleted because the simulation is a loss, then, only the reply G is kept. During the third simulation new replies are learned and added to the reply G. Learning is the same for white replies.

3.4 Last-Good-Reply

The Last-Good-Reply improvement [BD10] is similar to a plain PoolRave: the goal is to try to have a kind of learning in the playouts. Here, the RAVE moves are not used, but instead, the principle is to learn how to reply to a previous move. During the simulation, each reply to a move for a player is kept in memory. The reply is considered as a success if the simulation is a win. The Last-Good-Reply improvement can be improved with the addition of forgetting: all stored replies made by a losing simulation are deleted. This modification is called the Last-Good-Reply Forgetting -1 (LGRF-1, see Fig. 5).

There exist other versions of the Last-Good-Reply improvement, namely LGRF-2, LGR-1 and LGR-2. LGRF-2 is the same as LGRF-1 excepted that it considers the last two previous moves. LGR-1 and LGR-2 correspond to LGRF-1 and LGRF-2 with no forgetting. In this study, we only use LGRF-1 in our experiments because it works better for our player. This result is similar to the results found in [SWU12].

4 Comparison of MCTS-based methods

As seen in the previous section, RAVE improves the efficiency of MCTS by introducing some bias in the selection step. PoolRave and LGRF-1 extend this idea by introducing some bias in the simulation step, i.e., the Monte-Carlo part. In this section, we introduce a protocol (4.1), and we evaluate the efficiency of PoolRave and LGRF-1 for the application games, Hex and Havannah (4.2 and 4.3).

4.1 Protocol

To evaluate the efficiency of PoolRave/LGRF-1, we compare two algorithms (MCTS+RAVE+PoolRave and MCTS+RAVE+LGRF-1) to a baseline. The baseline is a MCTS player with the RAVE improvement. We have chosen this baseline (and not just a vanilla MCTS) because MCTS+RAVE is now considered as the standard algorithm, particularly for the game of Hex and for the game of Havannah. Thus the difference between the baseline and the improved algorithms is only the Monte-Carlo policy. In order to have a low standard deviation, we run many games (600 games at least for each experiment) and compute the win rate

Table 1: Hex: PoolRave/LGRF-1 vs Baseline. Results with board size = 11, $R = 90$ and $C = 0.4$.

player	playouts	pool proba	pool sim	pool size	win rate	std dev	nb games
PoolRave	1,000	0.1	10	30	48.25%	±1.11	2,000
		0.1	5	30	46.85%	±1.11	
		0.5	10	30	46.85%	±1.11	
		0.5	5	30	43.9%	±1.10	
		0.9	10	30	46.7%	±1.11	
		0.9	5	30	48.85%	±1.11	
	10,000	0.1	10	30	44.2%	±1.11	600
		0.5	10	30	41.95%	±1.10	
		0.9	10	30	43.95%	±1.10	
LGRF-1	1,000	-	-	-	51.9%	±1.11	2,000
	10,000	-	-	-	53.33%	±2.03	600

Table 2: Havannah: PoolRave/LGRF-1 vs Baseline. Results with board size = 8, $R = 90$ and $C = 0.4$.

player	playouts	pool proba	pool sim	pool size	win rate	std dev	nb games
PoolRave	1,000	0.1	10	30	69.15%	±1.03	2,000
		0.1	5	30	69.3%	±1.03	
		0.5	10	30	68.95%	±1.03	
		0.5	5	30	68.6%	±1.03	
		0.9	10	30	59.75%	±1.09	
		0.9	5	30	59.7%	±1.09	
	10,000	0.1	10	10	40.66%	±2.00	600
		0.5	10	10	37.5%	±1.97	
		0.9	10	10	36.66%	±1.96	
LGRF-1	1,000	-	-	-	57.95%	±1.11	2,000
	10,000	-	-	-	48.83%	±2.03	600

of the PoolRave/LGRF-1 player. Since playing the first move can be an advantage, we run half the games with PoolRave/LGRF-1 as the first player and the other games with PoolRave/LGRF-1 as the second player.

4.2 Hex

We have applied the previous protocol to the game of Hex and obtained the results depicted in Table 1. With the various parameter values we have tested, the probability that PoolRave wins against the baseline stands between 41% and 49%, i.e., PoolRave performs worse than the baseline. For LGRF-1, the win rate stands between 51% and 54%, i.e., LGRF-1 performs slightly better than the baseline. This indicates that, for a game such as Hex, biasing the Monte-Carlo policy step tends to be difficult.

4.3 Havannah

The results for the game of Havannah are depicted in Table 2. With a small number of random simulations ($1,000$ playouts), biasing the simulation step is very interesting: PoolRave wins 59% to 69% of the games against the baseline, and LGRF-1 about 58%. However, with a higher number of random simulations ($10,000$ playouts), biasing is not as interesting: PoolRave's win rate drops below 41% and LGRF-1's win rate below 49%. In fact, the number of playouts has a huge importance on the efficiency of the improvement, as shown in Fig. 6. We may notice that the advantage of the PoolRave enhancement decreases as the number of simulations increases. It even becomes negative above roughly $1,500$ simulations, which is quite small.

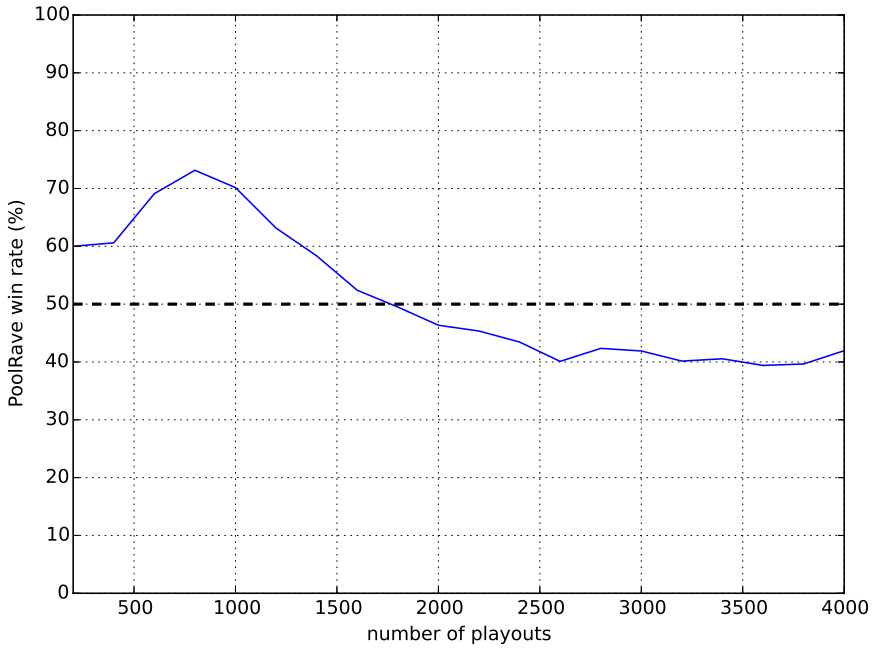

Figure 6: Havannah: PoolRave vs Baseline using different number of playouts. Results with nb games = 2000, board size = 8, $R = 90$, $C = 0.4$, pool proba = 0.1, pool sim = 10 and pool size = 10. With a small number of playouts, PoolRave is beneficial, but for $1,500$ playouts or more it becomes harmful.

5 Influence of game rules

Both Hex and Havannah belong to the family of connection games. They are quite related, and generally, a good human player at one of these games is also good at the other one. In this section, we try to understand why an improvement such as PoolRave works on Havannah but not on Hex. In 5.1 we discuss winning shapes that win effectively. In 5.2 we modify the rules of the game of Havannah. In 5.3 we perform new experiments for studying the behaviour of PoolRave/LGRF-1 on the modified games. The game of Havannah is interesting because it has both tactical and strategic elements. Therefore, as described in 5.4, modyfing the Havannah rules brings an interesting range of new games (some of them are even close to the game of Hex). Indeed, in the game of Havannah, the tactical short-term threats (rings) allow a win but forks are considered by experts as a more strategic long-term win.

5.1 Winning shapes that win effectively

As seen previously, to win a game of Havannah, a player has to realise a winning shape: ring, fork or bridge. In the previous section, we show that PoolRave plays better than the baseline when the number of playouts is small but not when the number of playouts is high (see Fig. 6).

To develop this analysis, we detail which winning shape the winner realised effectively (see Fig. 7). When the number of playouts is small, both PoolRave and the baseline mostly win by realising rings. This can be explained by the fact that, when the number of playouts is small, it is difficult for MCTS-based algorithms to detect that a ring is about to be realised. Since a ring can be realised with a few moves, such algorithms tend to play these moves.

When the number of playouts is high, PoolRave and the baseline win most of the time by realising forks. Indeed, the algorithms have now more “time” to evaluate moves. Thus, they can detect that the opponent player is about to realise a ring and they can evaluate more accurately the benefit of forks.

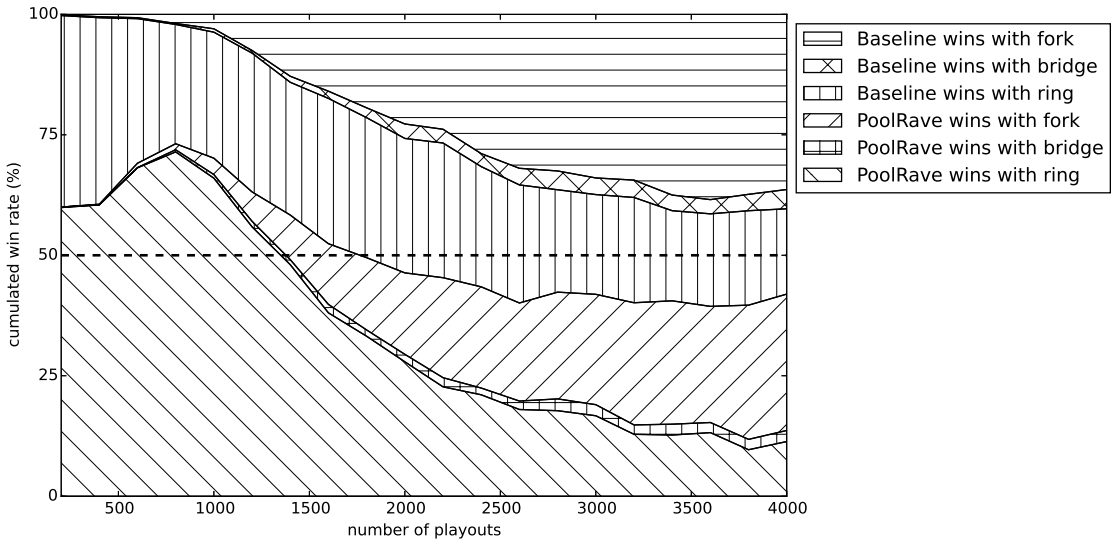

Figure 7: Havannah: PoolRave vs Baseline. Results with nb games = 2,000, board size = 8, $R = 90$, $C = 0.4$, pool proba = 0.1, pool sim = 10 and pool size = 10. With only few playouts, both the baseline and PoolRave win with ring, which is far from a real game. When the number of playouts increases, wins are mainly forks.

These results confirm the classic observations made by Havannah experts:

- forks are the most interesting strategies (long-term victories), and
- rings are interesting to do tactical threats (short-term profits).

This may also explain why the algorithms with a biased playout policy, such as PoolRave, become worse than the baseline when the number of simulations is large. Since these algorithms tend to make tactical threads, with a small number of playouts some of these threads are successful (they are not detected by the baseline). With a large number of playouts, the baseline can easily detect the threads so the biased playouts are not beneficial anymore. However, the biased algorithms still try to make tactical threads so they dedicate less playouts to elaborate a strategy than the baseline, hence the worse results.

5.2 Combining the winning shapes

Below, we design seven new games in order to find a relation between the winning shapes of Havannah and the benefits of an improvement of the Monte-Carlo policy. Indeed, we can easily change the rules by deleting the possibility of winning with one (or two) of the shapes. Thus, we have defined the following games:

- RBF (Ring, Bridge and Fork), which corresponds to the game of Havannah: a player has to realise a ring, a bridge or a fork in order to win,
- R, which corresponds to the game of Havannah without bridge and fork: a player has to realise a ring in order to win,
- B: a player has to realise a bridge,
- F: a player has to realise a fork,
- RF: a player has to realise a ring or a fork,
- RB: a player has to realise a ring or a bridge, and

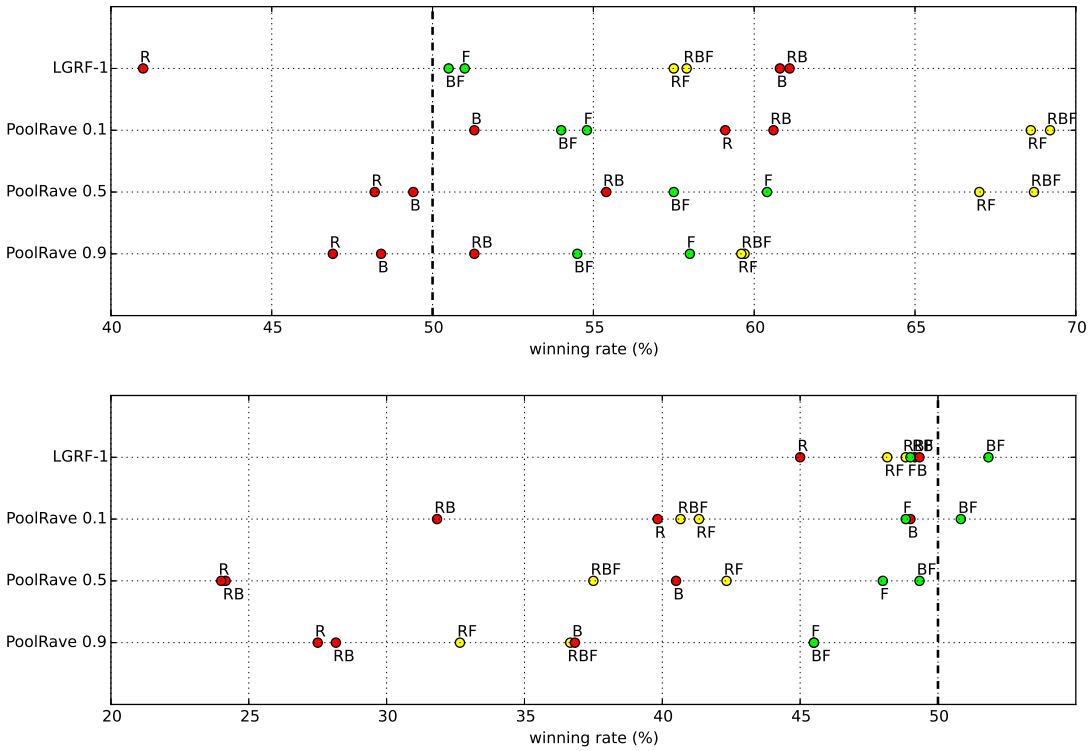

Figure 8: PoolRave/LGRF-1 vs Baseline on various Havannah-based games. Results with board size = 8, $R = 90$, $C = 0.4$, pool proba $\in \{0.1, 0.5, 0.9\}$, pool sim = 10 and pool size = 10. Top: nb playouts = $1,000$ and nb games = $2,000$. Bottom: nb playouts = $10,000$ and nb games = 600. The colors indicate the type of the games: tactical (red), strategic (green), tactical and strategic (yellow).

- BF: a player has to realise a bridge or a fork.

All other rules remain the same.

5.3 Experiments and results

We performed a set of experiments by using all these "new games" in order to evaluate the benefits of two Monte-Carlo policy improvements (PoolRave and LGRF-1). The goal is to see whether we can find some similarities or logic among these games. Since the benefits of Monte-Carlo policy improvements really depend on the number of playouts, we run experiments with both a small and a large number of playouts (see Fig. 8).

With $1,000$ playouts (see Fig. 8, top), the PoolRave player is better than the baseline on the games RBF and RF (PoolRave 0.5 win rate $> 65\%$). Indeed, PoolRave plays short-term tactics (rings) and the baseline does not have sufficient time to detect them since it also investigates long-term strategies (forks). On strategy-oriented games such as F and BF, the benefits of PoolRave are smaller (win rate about 60%). Here, it turns out that there is no short-term tactics to find, but a *biased* Monte-Carlo policy makes the algorithm find a quite good strategy sooner. On purely tactical games such as R (and maybe RB and B), it is not easy to find a long-term strategy, which makes both players look for short-term tactics in such a way that no one has a real advantage (most win rates between 45% and 55%).

With $10,000$ playouts (see Fig. 8, bottom), PoolRave plays worse than the baseline but this depends on the game. On RBF and RF (both tactical and strategic games), PoolRave still spends time to evaluate tactical threats (rings) but the baseline has now sufficient playouts to detect them. Since the baseline is

more strategy-oriented (which is most promising for such Havannah-like games), biasing the Monte-Carlo policy brings no benefit (PoolRave 0.5 win rate about 40%). On strategy-oriented games such as F and BF, PoolRave does not lose time to evaluate useless short-term tactic (ring), therefore it is almost as good as the baseline. On tactical games (R, RB and B), PoolRave is really inefficient (on R, PoolRave 0.5 win rate $< 25\%$ and draw rate $= 25\%$) because it favours exploitation over exploration, which is not interesting for such short-term games.

We may notice that LGRF-1 is more robust to the various games and to the number of playouts, thanks to its forgetting trend. Thus, its gain with $1,000$ playouts is smaller but its loss with $10,000$ playouts is also smaller. However, like PoolRave, LGRF-1 is inefficient on R (with $1,000$ playouts: win rate $\approx 40\%$; with $10,000$ playouts: win rate $\approx 45\%$ and draw rate $\approx 30\%$) since the baseline performs quite well on this new game, for which tactical goals are essential.

5.4 Discussion

Our results show that improving the Monte-Carlo policy can yield benefits for small numbers of playouts; this is consistent with previous results obtained for Havannah [SWU12] and for Go [BD10]. However, our results also show that improving the Monte-Carlo policy is beneficial only if the game can be won with both a short-term tactic and a long-term strategy. Indeed, when the number of playouts is large, the tactical threads favoured by biased Monte-Carlo policies are not beneficial anymore since they can easily be detected. Moreover, if the game has only strategic elements (or only tactical elements), there is no choice to make between a tactical or a strategic move, which reduces the possible benefit of the biased Monte-Carlo policies.

This seems to confirm our insight that the biased Monte-Carlo policies do not perform very well in Hex because there are no obvious tactical short-term goals, but only long-term wins, involving longer (global) chains in complex configurations. In Havannah, the rings tend to involve smaller, simpler (local) chains which correspond to tactical threats, and those are more likely to be picked up correctly by the biased versions of MCTS.

Finally, we may notice that it is difficult to classify a game as "tactical" or "strategic" (or both), since these notions are more appropriate to characterise a player. The games RBF and RF can be classified as tactical and strategic since a player can effectively make tactical or strategic moves. With the games F and BF, there are no obvious tactical moves leading to a short-term win, so these games can be classified as strategic. However, the games R, RB and B are more difficult to classify because rings and bridges, which were previously seen as tactical threads, are now the only possible winning shapes; so, they can be considered here as "short-term strategies". This may also explain the great variability of the results for these games, in our experiments.

6 Conclusion

In this article, we studied the impact of biasing the Monte-Carlo policy of the Monte-Carlo Tree Search. To this end, we compared an MCTS algorithm with an unbiased Monte-Carlo policy and an MCTS algorithm with a biased Monte-Carlo policy. We used the classic RAVE improvement (biased tree policy) for all algorithms. For the MCTS algorithm with a biased Monte-Carlo policy, we applied two biasing techniques: PoolRave and LGRF-1.

We did experiments with these algorithms on two connection games, the game of Hex and the game of Havannah. First, we noticed that the algorithms with a biased Monte-Carlo policy (especially the PoolRave improvement) are more efficient on Havannah than on Hex. Subsequently, we studied for Havannah, how the benefits of the biased algorithms are related to the number of random simulations (playouts).

Then, we compared the ability of the algorithms to achieve short-term tactical goals and long-term strategic goals, by modifying the game of Havannah. We found that, when the number of playouts is small, the biased algorithms are beneficial and generally win by realising a ring, which corresponds to a short-term tactic. However, when the number of playouts is high, they become harmful and the most interesting shape is a fork, corresponding to a long-term strategy. Indeed, the (unbiased) opponent has then sufficient playouts

to detect short-term tactics. So, dedicating playouts to find tactics is not only useless but also reduces the number of playouts to find interesting strategies.

It is really hard to make general conclusions, with experiments based on only two games. Still, we hope that our study will form a basis for new insights into MCTS and its enhancements. For example, since all the improvements presented in this article have been experimented in the game of Go, it would be interesting to do an analogous study in order to understand why one improvement is better than another one for this game.

Further work is needed to investigate the claim stated in the paper, viz. that improving the MC policy can yield benefits for a small number of playouts. In particular further work is needed to investigate whether biasing the Monte-Carlo policy increases the exploitation of tactical moves, which is beneficial when the (unbiased) opponent does not have sufficient playouts to detect them. We remark that with a higher number of playouts, such moves are easy to detect, so it is more fruitful to explore the area for strategic moves.

It would also be interesting to study the "decisive moves" improvement proposed in [TT10]. There it was shown that the improvement would detect the biggest tactical threats, making biased policies more effective.

As a second future work, it would be interesting to compare the biased algorithms mentioned above with non MCTS-based algorithms, or expert human players to avoid the possible negative effects of self-playing.

References

[AHH09] Broderick Arneson, Ryan Hayward, and Philip Henderson. MoHex wins Hex tournament. *International Computer Games Association (ICGA) Journal*, 32(2):114–116, 2009.

[AHH10] 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.

[BCC+10] Amine Bourki, Guillaume Chaslot, Matthieu Coulm, Vincent Danjean, Hassen Doghmen, Jean-Baptiste Hoock, Thomas Hérault, Arpad Rimmel, Fabien Teytaud, Olivier Teytaud, Paul Vayssière, and Ziqin Yut. Scalability and parallelization of monte-carlo tree search. In *7th Computers and Games Conference, Kanazawa, Japan*, volume 6515, pages 48–58. Springer, Heidelberg, 2010.

[BD10] H. Baier and P.D. Drake. The power of forgetting: Improving the last-good-reply policy in Monte-Carlo Go. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(4):303–309, 2010.

[Ber76] David Berman. Hex must have a winner: An inductive proof. *Mathematics Magazine*, 49(2):85–86, 1976.

[BPW+12] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I 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, 2012.

[Caz09] Tristan Cazenave. Monte-Carlo Kakuro. In *12th Advances in Computer Games Conference, Pamplona, Spain*, volume 6048, pages 45–54. Springer, Heidelberg, 2009.

[CFH+10] Guillaume Chaslot, Christophe Fiter, Jean-Baptiste Hoock, Arpad Rimmel, and Olivier Teytaud. Adding expert knowledge and exploration in Monte-Carlo Tree Search. In *7th Computer and Games Conference, Kanazawa, Japan*, pages 1–13. Springer, Heidelberg, 2010.

[CHTT09] G. Chaslot, J.B. Hoock, F. Teytaud, and O. Teytaud. On the huge benefit of quasi-random mutations for multimodal optimization with application to grid-based tuning of neurocontrollers. In *European Symposium on Artificial Neural Networks*, 2009.

[Cou07] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo Tree Search. In *5th Computers and Games Conference, Turin, Italy*, volume 4630, pages 72–83. Springer, Heidelberg, 2007.

[CSB+06] GMJB Chaslot, Jahn-Takeshi Saito, Bruno Bouzy, JWHM Uiterwijk, and H Jaap Van den Herik. Monte-Carlo strategies for computer Go. In *BeNeLux Conference on Artificial Intelligence*, pages 83–91, 2006.

[Ewa12] Timo Ewalds. Playing and solving Havannah. Master's thesis, University of Alberta, 2012.

[Fin12] Hilmar Finnsson. Generalized monte-carlo tree search extensions for general game playing. In *AAAI Conference on Artificial Intelligence*, 2012.

[Gal79] David Gale. The game of Hex and the Brouwer Fixed-Point theorem. *The American Mathematical Monthly*, 86(10):818–827, 1979.

[GS07] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *International Conference on Machine Learning*, pages 273–280, 2007.

[HS14] Johannes Heinrich and David Silver. Self-play monte-carlo tree search in computer poker. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[KS06] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

[Lor08] Richard J Lorentz. Amazons discover Monte-Carlo. In *6th Computers and Games Conference, Beijing, China*, volume 5131, pages 13–24. Springer, Heidelberg, 2008.

[Lor10] Richard J. Lorentz. Improving Monte-Carlo Tree Search in Havannah. In *7th Computers and Games Conference, Kanazawa, Japan*, volume 6515, pages 105–115. Springer, Heidelberg, 2010.

[Maa05] Thomas Maarup. Hex everything you always wanted to know about hex but were afraid to ask. Master's thesis, University of Southern Denmark, 2005.

[NM09] Hootan Nakhost and Martin Müller. Monte-Carlo exploration for deterministic planning. In *International Jont Conference on Artifical Intelligence*, pages 1766–1771, 2009.

[RTT10] A. Rimmel, F. Teytaud, and O. Teytaud. Biasing Monte-Carlo simulations through rave values. In *7th Computers and Games Conference, Kanazawa, Japan*, volume 6515, pages 59–68. Springer, Heidelberg, 2010.

[Sch92] R.W. Schmittberger. *New Rules for Classic Games*. John Wiley & Sons Inc, 1992.

[Stu15] Nathan R Sturtevant. *Monte Carlo Tree Search and Related Algorithms for Games*. CRC Press, 2015.

[SWU12] Jan A Stankiewicz, Mark HM Winands, and Jos WHM Uiterwijk. Monte-Carlo Tree Search enhancements for Havannah. In *13th Advances in Computer Games Conference, Tilburg, The Netherlands*, volume 7168, pages 60–71. Springer, Heidelberg, 2012.

[TT09] F. Teytaud and O. Teytaud. Creating an upper-confidence-tree program for Havannah. In *12th Advances in Computer Games Conference, Pamplona, Spain*, volume 6048, pages 65–74. Springer, Heidelberg, 2009.

[TT10] F. Teytaud and O. Teytaud. On the huge benefit of decisive moves in monte-carlo tree search algorithms. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 359–364. IEEE, 2010.

[WM14] Karol Waledzik and Jacek Mandziuk. An automatically generated evaluation function in general game playing. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(3):258–270, 2014.