# Monte-Carlo Tree Search Enhancements for Havannah

Jan A. Stankiewicz, Mark H.M. Winands, and Jos W.H.M. Uiterwijk

Department of Knowledge Engineering, Maastricht University
j.stankiewicz@student.maastrichtuniversity.nl,
{m.winands,uiterwijk}@maastrichtuniversity.nl

**Abstract.** This article shows how the performance of a Monte-Carlo Tree Search (MCTS) player for Havannah can be improved by guiding the search in the playout and selection steps of MCTS. To improve the *playout* step of the MCTS algorithm, we used two techniques to direct the simulations, Last-Good-Reply (LGR) and N-grams. Experiments reveal that LGR gives a significant improvement, although it depends on which LGR variant is used. Using N-grams to guide the playouts also achieves a significant increase in the winning percentage. Combining N-grams with LGR leads to a small additional improvement. To enhance the *selection* step of the MCTS algorithm, we initialize the visit and win counts of the new nodes based on pattern knowledge. By biasing the selection towards joint/neighbor moves, local connections, and edge/corner connections, a significant improvement in the performance is obtained. Experiments show that the best overall performance is obtained when combining the visit-and-win-count initialization with LGR and N-grams. In the best case, a winning percentage of 77.5% can be achieved against the default MCTS program.

## 1 Introduction

Recently a new paradigm for game-tree search has emerged, the so-called Monte-Carlo Tree Search (MCTS) [6,13]. It is a best-first search algorithm that is guided by Monte-Carlo simulations. In the past few years MCTS has substantially advanced the state-of-the-art in several deterministic game domains where $\alpha\beta$-based search [12] has had difficulties, in particular computer Go [15], but other domains include General Game Playing [3], LOA [25] and Hex [1]. These are all examples of game domains where either a large branching factor or a complex static evaluation function do restrain $\alpha\beta$ search in one way or another.

A game that has recently caught the attention of AI researchers is Havannah, regarded as one of the hardest connection games for computers [24]. Designing an effective evaluation function is quite hard and the branching factor is rather large, making MCTS the algorithm of choice. A substantial amount of research has been performed for applying MCTS in Havannah [16,19,23,24], but humans are still superior. In this article[1] we therefore investigate how the performance

[1] This article is based on the research performed by the first author for his M.Sc. thesis [21].

H.J. van den Herik and A. Plaat (Eds.): ACG 2011, LNCS 7168, pp. 60–71, 2012.

of our MCTS-based Havannah program [8,11,21] can be improved by enhancing the playout and selection steps. For the playout step we propose to apply the Last-Good-Reply policy [2,7] and N-grams [14,20]. For the selection step, we bias the moves by using prior knowledge [10] based on patterns.

The article is organized as follows. In Section 2 we explain the rules of Havannah. Next, Section 3 discusses the application of MCTS to Havannah and describes our enhancements for the playout and selection steps. Subsequently, the enhancements are empirically evaluated in Section 4. Finally, in Section 5 we conclude and give an outlook on future research.

## 2 The Rules of Havannah

Havannah is a turn-based two-player deterministic perfect-information connection game invented by Christian Freeling in 1976 [9]. It is played on a hexagonal board, often with a *base* of 10, meaning that each side has a length of 10 cells. One player uses white stones; the other player uses black stones. The player who plays with white stones starts the game. Each turn, a player places one stone of his color on an empty cell. The goal is to form one of the following three possible winning connections (also shown in Fig. 1).

- **Bridge:** A connection that connects any two corner cells of the board.
- **Fork:** A connection that connects three sides. Corner cells do not count as side cells.
- **Ring:** A connection that surrounds at least one cell. The cell(s) surrounded by a ring may be empty or occupied by white or black stones.

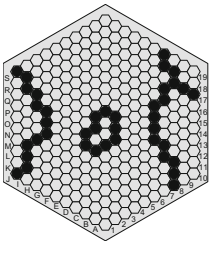

**Fig. 1.** The three possible connections to win the game. From left to right: a bridge, a ring and a fork.

Because White has an advantage being the starting player, the game is often started using the swap rule. One of the players places a white stone on the board after which the other player may decide whether he[2] will play as White or Black. It is possible for the game to end in a draw, although this is quite unlikely.

[2] For brevity, we use 'he' and 'his' whenever 'he or she' and 'his or her' are meant.

## 3 Havannah and Monte-Carlo Tree Search

MCTS is a best-first search algorithm that combines Monte-Carlo evaluation (MCE) with tree search [6,13]. We assume the MCTS algorithm to be known by the readers. For more details we refer to the literature, notably the Ph.D. thesis by Chaslot [4]. In this section we first describe previous MCTS research related to Havannah (3.1), then give our MCTS enhancements for the play-out (3.2) and selection (3.3) steps.

### 3.1 MCTS Refinements for Havannah

Teytaud and Teytaud [24] introduced MCTS based on UCT in Havannah. Their experiments showed that the number of play-outs per move has a significant impact on the performance. Additions such as Progressive Widening [5] and Rapid Action Value Estimates (RAVE) [10] were used as well. The former gave a small improvement in the winning rate, while RAVE significantly increased the percentage of games won. An enhancement of RAVE (called PoolRave) applied to Havannah gave a further small increase in the winning rate [19]. The idea of adding automatically generated knowledge in the play-out step to guide simulations was first explored by Rimmel and Teytaud [18]. This was dubbed *contextual Monte-Carlo* (CMC) simulation and was based on a reward function learned on a tiling of the simulation space. Experiments for Havannah showed a winning rate of 57% against a program without CMC.

More important was the application of *decisive moves* during the selection and play-out step: whenever there is a winning move, that move is played regardless of the other possible moves. Experiments showed winning percentages in the range of 80% to almost 100% [23]. Fossel [8] used Progressive History [17] and proposed Extended RAVE to improve the selection strategies, giving a winning percentage of approximately 60%.

Several more enhancements for an MCTS player in Havannah were discussed by Lorentz [16]. One is to try to find moves near stones already on the board, thus avoiding playing in empty areas. Another is to recognize forced wins and losses, called Havannah-Mate, which can save time during the search process. A third enhancement is the use of the Killer RAVE heuristic, where only the most important moves are used for computing RAVE values. Each of these enhancements caused a significant increase in the winning percentage.

### 3.2 Enhancing the Play-out Step in MCTS

This subsection discusses the Last-Good-Reply policy and N-grams which may improve the play-out step of MCTS in Havannah.

**Last-Good-Reply.** The Last-Good-Reply (LGR) policy [2,7] is an enhancement used during the play-out step of the MCTS algorithm. Rather than applying the default simulation strategy, moves are chosen according to the last good replies to previous moves, based on the results from previous play-outs.

It is often the case in Havannah that certain situations are played out locally. This means that if a certain move is a good reply to another move on some board configuration, it will likely be a good reply to that move on different board configurations as well, because it is only the local situation that matters. However, MCTS itself does not 'see' such similar local situations. The goal of LGR is to improve the way in which MCTS handles such local moves and replies.

There are several variants of LGR [2]. The first one is LGR-1, where each of the winner's moves made during the play-out step is stored as the last good reply to the previous move. During the play-out step of future iterations of the MCTS algorithm, the last good reply to the previous move is always played instead of a (quasi-)random move, whenever possible. Otherwise, the default simulation strategy is used. As an example, consider Fig. 2, where a sequence of moves made during a play-out is shown.

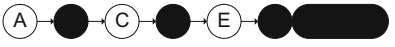


**Fig. 2.** A simulation in MCTS

Because Black is the winner, the LGR-1 table for Black is updated by storing every move by Black as the last good reply to the previous one. For instance, move B is stored as the last good reply to move A. If White will play move A in future play-outs, Black will always reply by playing B if possible.

The second variant of LGR is LGR-2. As the name suggests, LGR-2 stores the last good reply to the previous two moves. The advantage of LGR-2 is that the last good replies are based on more relevant samples [2]. During the play-out, the last good reply to the previous *two* moves is always played whenever possible. If there is no last good reply known for the previous two moves, LGR-1 is tried instead. Therefore, LGR-2 also stores tables for LGR-1 replies.

A third variant is LGR-1 with forgetting, or simply LGRF-1. This works exactly the same as LGR-1, but now the loser's last good replies are deleted if they were played during the play-out. Consider Fig. 2 again, where White lost the game. For instance, if move C was stored as the last good reply to B for White, it is deleted. Thus, the next time Black plays B, White will chose a move according to the default simulation strategy.

The fourth and last variant is LGRF-2, which is LGR-2 with forgetting. Thus, the last good reply to the previous two moves is stored and after each play-out, the last good replies of the losing player are deleted if they have been played.

**N-grams.** The concept of N-grams was originally developed by Shannon [20], where he discussed how one can predict the next word, given the previous $N-1$ words. Typical applications of N-grams are, e.g., speech recognition and spelling checkers, where the previous words spoken or written down can help to determine what the next word should be. However, N-grams are also applicable in the context of games, as shown by Laramée [14]. They can be used as an enhancement to the play-out step of the MCTS algorithm. The idea is somewhat similar to LGR. Again, moves are chosen according to their predecessor, but instead of

choosing the last successful reply, the move with the highest winning percentage so far among all legal moves is chosen. Thus, for each legal move $i$, the ratio $w_{i,j}/p_{i,j}$ is calculated, where $w_{i,j}$ is the number of times playing move $i$ in reply to move $j$ led to a win and $p_{i,j}$ is the number of times move $i$ was played in reply to move $j$. In order not to make the search too deterministic, the moves are chosen in an $\epsilon$-greedy manner [22]. With a probability of $1 - \epsilon$ an N-gram move is chosen, while in all other cases, a move is chosen based on the default simulation strategy. Furthermore, the values $w_{i,j}$ and $p_{i,j}$ in the N-gram tables are multiplied by a decay factor $\gamma$ after every move played in the game, where $0 \leq \gamma \leq 1$. This ensures that, as the game progresses, new moves will be tried as well, instead of only playing the same N-gram moves over and over again.

It is also possible to combine N-grams with a certain threshold $T$. The reason to apply thresholding, is to try to improve the reliability of N-grams. The more often a certain N-gram has been played, the more reliable it is. If the N-gram of a proposed move has been played fewer than $T$ times before, the move is not taken into consideration. If all of the available moves have been played fewer than $T$ times, the default simulation strategy is applied.

Like with LGR, N-grams can be extended to take into account the previous two moves, instead of only the previous move. To distinguish between the two, 'N-gram1' refers to N-grams based on only the previous move, while 'N-gram2' refers to N-grams based on the previous two moves.

Because N-gram1 and N-gram2 are based on different contexts, combining the two may give a better performance than using N-gram1 or N-gram2 separately. N-gram1 and N-gram2 can be combined using averaging. Rather than choosing moves based purely on N-gram2, the moves are chosen based on the average of the ratios $w_{i,j}/p_{i,j}$ of N-gram1 and N-gram2.

### 3.3 Enhancing the Selection Step in MCTS

Gelly and Silver [10] proposed the use of prior knowledge for the selection step of the MCTS algorithm, where the visit and win counts of the new nodes are initialized to certain values, based on the properties of the move to which the node corresponds. It is basically an adaptation of the UCT formula, as shown in Equation 1.

$$k \in argmax_{i \in I} \left( \frac{v_i n_i + \alpha_i}{n_i + \beta_i} + C \cdot \sqrt{\frac{\ln n_p}{n_i + \beta_i}} \right) \tag{1}$$

The additional parameters $\alpha_i$ and $\beta_i$ are the win count bias and visit count bias, respectively, which are based on the properties of the move corresponding to node $i$. By adding such biases to the win and visit counts of MCTS nodes, the selection can be biased towards certain moves.

This subsection discusses three heuristics how the values of $\alpha_i$ and $\beta_i$ can be chosen. First, we describe how the selection can be biased towards joint moves and neighbor moves. Then we discuss how local connections can be used to guide the selection. Finally, we describe how the selection can be biased towards edge and corner connections. See Fig. 3 for an overview.

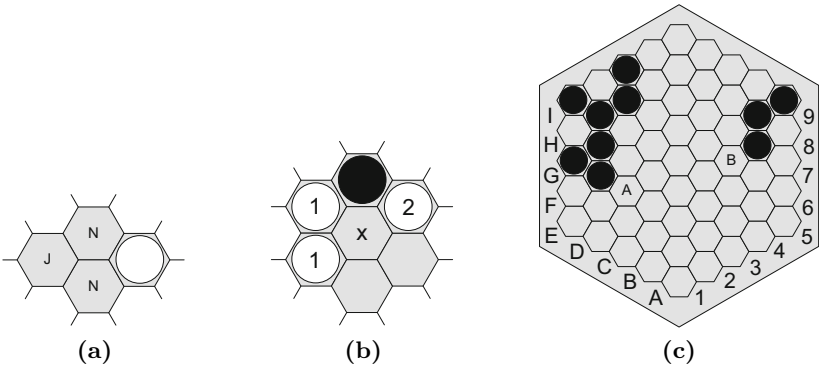


**Fig. 3.** Biasing the selection towards certain types of moves: joint and neighbor moves (a), local connections (b), and edge and corner connections (c)

**Joint and Neighbor Moves.** Lorentz [16] proposed to initialize the visit and win counts of new nodes in such a way that the selection is biased towards 'joint moves' and 'neighbor moves'. Joint moves are moves that are located two cells from a stone of the current player and where the two cells between are empty. An example is shown in Fig. 3a, where the joint move is marked by 'J', viewed from White's perspective. Neighbor moves are simply moves adjacent to a cell of the same color, marked by 'N' in Fig. 3a.

**Local Connections.** A second idea to initialize visit and win counts of new nodes is to take into account the number of connections with existing groups. After all, Havannah is a game in which connections play an important role. As an example, consider Fig. 3b. Assuming that White considers playing at cell '×', there are two local groups of white surrounding stones. They are indicated by the number of the group to which the stone belongs. The stones marked by '1' belong to the same group, as they are connected with each other. The stone marked by '2' is a separate group. If one would play move '×', it would thus form a connection between two local groups. Of course, it could be the case that the two groups are actually connected outside this local area, in which case playing move '×' would simply complete a ring.

**Edge and Corner Connections.** A third option is to count the number of edges and corners a proposed move would be connected to. The idea is to try to direct the search towards the formation of forks and bridges. For example, see Fig. 3c. Move 'A' on cell E3 connects to one corner and two edges, whereas move 'B' on cell D6 only connects to one corner. Move 'A' is therefore likely to be better than move 'B'. Our MCTS engine already keeps track to which chain each stone belongs [11]. It means that the only additional calculations are (1) to determine which chains the proposed move is connected to and (2) to check for edge and corner cells whether they belong to one of those chains as well.

## 4 Experiments

This section presents the experiments to assess the enhancements described in the previous section. After describing the experimental setup (4.1), we report the results of experiments with Last-Good-Reply (4.2) and N-grams (4.3). Finally, experiments with several visit-and-win-count initializations (4.4) are discussed.

### 4.1 Experimental Setup

The experiments discussed in this section were performed on a 2.4 GHz AMD Opteron CPU with 8 GB RAM. Both players use UCT/RAVE with UCT constant $C$ (see Equation (1)) set to 0.4 and RAVE constant $R$ (see [23]) set to 50, plus Havannah-Mate. The default simulation strategy is a uniform random playout. Every experiment was done in a self-play setup. Unless stated otherwise, each experiment consists of 1,000 games on a base-5 board, with a thinking time of 1 second. Because White has a starting advantage, each experiment is split into two parts. During the first 500 games, the enhancements investigated are applied to White and during the second 500 games, they are applied to Black. Throughout the experiments confidence intervals of 95% are used.

### 4.2 Last-Good-Reply

The performance of Last-Good-Reply was tested with the default setup described in Subsection 4.1. For this first set of experiments, the contents of the LGR tables of the previous turn were used as the initial LGR tables for the current move. The results for all four LGR variants are shown in Table 1.

**Table 1.** Performance of the four LGR variants

| | White | Black | Average |
|---|---|---|---|
| LGR-1 | 65.8% | 49.4% | 57.6% (±3.1) |
| LGR-2 | 62.4% | 47.8% | 55.1% (±3.1) |
| LGRF-1 | 65.8% | 58.0% | 61.9% (±3.0) |
| LGRF-2 | 59.0% | 49.6% | 54.3% (±3.1) |

As the table shows, LGR generally improves the performance of the MCTS engine. LGR-1 and LGR-2 seem to perform equally well given the confidence intervals, with winning percentages of 57.6% and 55.1%, respectively. Forgetting poor replies seems to give a slight improvement when added to LGR-1, but when added to LGR-2, the performance appears to be the same. LGRF-1 gives a winning percentage of 61.9% while LGRF-2 only wins 54.3% of the games.

Quite remarkably, LGR-2 and LGRF-2 do not perform better than LGR-1 and LGRF-1. In particular, the difference between the performance of LGRF-2 and LGRF-1 is quite significant. It appears that taking a larger context into account when using last good replies, does not lead to a better performance of the MCTS engine. As an additional experiment we tested whether emptying

the LGR table after every move would influence the performance. It turned out, however, that resetting the tables does not have any influence on the performance of the MCTS player. Each of the results lies within the confidence interval of its no-reset equivalent.

### 4.3 N-grams

N-grams were tested with the default setup described in Subsection 4.1. We first summarize three experiments for configuring the N-grams. For detailed results, see [21]. Next, we investigate the combination of N-grams with LGR policies.

**N-gram Configuration Experiments.** The first experiment was to determine the best value of the decay factor $\gamma$. The N-grams were chosen using $\epsilon$-greedy selection, with $\epsilon = 0.1$. As it turned out, the best value for $\gamma$ appears to be 0, which means that the N-gram tables are completely reset for each turn. The resulting winning percentages for N-gram1 and N-gram2 are $60.2 \pm 3.0\%$ and $61.3 \pm 3.0\%$, respectively.

In the second set of experiments with N-grams the influence of thresholding was evaluated. Thus, a move was only considered if its N-gram had been played more than $T$ times before. In the case of N-gram2, if the N-gram was played fewer than $T$ times, the N-gram1 of the move was considered. A decay factor $\gamma = 0$ was used. It turned out that thresholding has no positive influence on the performance. In fact, the higher the threshold, the lower the performance seems to get for each of the N-gram variants.

The third set of experiments was performed to determine the influence of averaging N-gram1 and N-gram2, rather than using only N-gram2. Again, the experiment was run with $\gamma = 0$ for different threshold values. The result was that using the average of both N-grams instead of only N-gram2 generally does not give a significant improvement. Again, applying a threshold only decreases the performance. When no threshold is applied, the result is within the confidence interval of the 61.3% result (i.e., the first experiment).

**N-grams Combined With LGR.** The fourth set of experiments evaluated the performance when N-grams are combined with Last-Good-Reply. The moves in the play-out are chosen as follows. First, the last good reply is tried. If none exists or if it is illegal, the move is chosen using N-grams with a probability of 0.9, thus $\epsilon = 0.1$. Otherwise, the default simulation strategy is applied. Again, the decay factor was set to $\gamma = 0$. No thresholding or averaging was applied to the N-grams. The results are shown in Table 2.

As the table shows, combining LGR with N-grams gives overall a better result. Given the confidence intervals, there seems to be little difference between the performances of the different combinations of LGR and N-grams. For the remainder of the experiments we choose the combination LGRF-2 with N-gram1.

**Table 2.** The winning percentages of N-grams with and without LGR variants

| | N-gram1 | N-gram2 |
|---|---|---|
| no LGR | 60.2% (±3.0) | 61.3% (±3.0) |
| LGR-1 | 62.0% (±3.0) | 65.0% (±3.0) |
| LGR-2 | 61.0% (±3.0) | 60.2% (±3.0) |
| LGRF-1 | 62.9% (±3.0) | 65.6% (±2.9) |
| LGRF-2 | 65.9% (±2.9) | 62.2% (±3.0) |

### 4.4 Initializing Visit and Win Count

To test the performance when visit-and-win-count initialization is used in the selection step of MCTS, four sets of experiments were constructed. All results are summarized in Table 3 at the end of this subsection.

**Joint and Neighbor Moves.** The influence of biasing the selection towards joint and neighbor moves was tested using the default setup, with the required parameters set as follows. All new nodes were initialized with a visit count of 40. Joint moves were given a win count of 30, neighbor moves a win count of 40, and all other moves a win count of 5.

Biasing the selection towards joint and neighbor moves improves the performance considerably, with a winning percentage of 69.2%, which is close to the 67.5% that Lorentz [16] achieved. The experiment was repeated with the addition of LGRF-2 and N-gram1 to the play-out step, increasing the performance significantly to 77.5%. The idea of biasing towards joint moves and neighbor moves was also extended to the play-out step. However, it turned out that this decreases the performance. The reason for this decrease in performance is most likely the computational cost of determining whether a cell corresponds to a joint or neighbor move.

**Local Connections.** The experiments with biasing the selection towards local connections, were run using the default setup, with the parameter values set as follows. All nodes were initialized with a visit count of 30. Nodes of which the corresponding move was connected to 0, 1, 2, or 3 surrounding groups were given initial win counts of 0, 10, 20, and 30, respectively.

For this initialization scheme, a winning percentage of 61.3% is achieved. This performance is somewhat less than when biasing towards joint and neighbor moves. Again, the experiments were repeated with the addition of LGRF-2 and N-gram1, increasing the winning percentage to 70.0%. However, biasing the selection towards joint and neighbor moves still performs significantly better.

**Edge and Corner Connections.** A third set of experiments was performed with biasing the selection towards edge and corner connections. For these experiments, the initial visit and win counts were set as follows. If a proposed move would be connected to more than 1 corner or more than 2 edges, the initial visit and win counts were set to 1 and 1000, respectively. In all other cases, the initial

visit count was set to 30, while the initial win count was set to 10 times the number of edges or corners to which the proposed move would be connected.

For this scheme, a winning percentage of 58.7% is achieved, slightly smaller than biasing towards joint and neighbor moves or local connections. When LGRF-2 and N-gram1 is added, the performance increased significantly to 68.3%.

**Combination.** The fourth set of experiments with visit-and-win-count initialization was based on a combination of the three heuristics described above. This was done in a cumulative way. The initial visit count for each node was set to 100, which is the combined initial visit count of the three heuristics. The initial win count was determined by combining the relevant initial visit counts of the three heuristics. Nodes of which the move would be connected to more than 1 corner or 2 edges, were again given an initial visit count of 1 and a win count of 1000. Combining the three heuristics gives good results (73.4% winning percentage). The combination works better than any of the three heuristics on their own. Adding LGRF-2 and N-gram1 again increases the performance, although the increase is not as large as with the three heuristics individually. In fact, the addition of LGRF-2 and N-gram1 performs just as well as the first heuristic, where the selection is biased towards joint and neighbor moves (77.5%).

**Table 3.** Biasing the selection towards certain types of moves, without and with the addition of LGRF-2 and N-gram1

| | without LGRF-2 + N-gram1 | with LGRF-2 + N-gram1 |
|---|---|---|
| joint and neighbour moves | 69.2% (±2.9) | 77.5% (±2.6) |
| local connections | 61.3% (±3.0) | 70.0% (±2.8) |
| edge and corner connections | 58.7% (±3.1) | 68.3% (±2.9) |
| combination | 73.4% (±2.7) | 77.5% (±2.6) |

## 5 Conclusions and Future Research

In this article we investigated for the game of Havannah several enhancements in the play-out and selection step of MCTS. Based on the experimental results we offer three conclusions. The first conclusion we may draw is that by adding LGR and N-grams to the *play-out step* of MCTS a large performance gain is achieved, both when these two enhancements are used separately or in combination with each other. Especially, LGRF-2 and N-gram1 seem to be a strong combination.

The second conclusion we may give is that by using pattern knowledge to initialize the visit and win counts of the new nodes, the *selection step* is considerably enhanced. By biasing the selection towards joint/neighbor moves, local connections and edge/corner connections, a significant improvement in the playing strength of the MCTS program is observed.

For the third conclusion we may state that the best overall performance is achieved when visit-and-win-count initialization is combined with LGRF-2 and N-gram1. Experiments reveal a winning percentage of 77.5%.

There are several directions for future research. The first potential improvement is tweaking the parameter values for visit-and-win-count initialization. Furthermore, the combination of the three visit-and-win-count initialization heuristics may be improved. One may consider altering the importance of each of the three heuristics within the combination. A second idea is to let the importance of each of the three heuristics be dynamic with respect to the current stage of the game. For example, during the first stages of the game, one could bias the selection only towards joint and neighbor moves, because there are almost no chains yet on the board. As the game progresses and chains are formed, the importance of local and edge/corner connections may be increased while that of joint and neighbor moves is decreased.

**Acknowledgments.** We gratefully acknowledge earlier work on our Havannah-playing agent by Bart Joosten and Joscha-David Fossel as reported in their B.Sc. theses [11,8].

## References

1. Arneson, B., Hayward, R.B., Henderson, P.: Monte Carlo Tree Search in Hex. IEEE Transactions on Computational Intelligence and AI in Games 2(4), 251–258 (2010)
2. Baier, H., Drake, P.D.: The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte Carlo Go. IEEE Transactions on Computational Intelligence and AI in Games 2(4), 303–309 (2010)
3. Björnsson, Y., Finnsson, H.: CadiaPlayer: A Simulation-Based General Game Player. IEEE Transactions on Computational Intelligence and AI in Games 1(1), 4–15 (2009)
4. Chaslot, G.M.J.-B.: Monte-Carlo Tree Search. PhD thesis, Maastricht University, Maastricht, The Netherlands (2010)
5. Chaslot, G.M.J.-B., Winands, M.H.M., Uiterwijk, J.W.H.M., van den Herik, H.J., Bouzy, B.: Progressive Strategies for Monte-Carlo Tree Search. New Mathematics and Natural Computation 4(3), 343–357 (2008)
6. Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M(J.) (eds.) CG 2006. LNCS, vol. 4630, pp. 72–83. Springer, Heidelberg (2007)
7. Drake, P.D.: The Last-Good-Reply Policy for Monte-Carlo Go. ICGA Journal 32(4), 221–227 (2009)
8. Fossel, J.D.: Monte-Carlo Tree Search Applied to the Game of Havannah. Bachelor's thesis, Maastricht University, Maastricht, The Netherlands (2010)
9. Freeling, C.: Introducing Havannah. Abstract Games 14, 14–20 (2003)
10. Gelly, S., Silver, D.: Combining Online and Offline Knowledge in UCT. In: Ghahramani, Z. (ed.) Proceedings of the 24th International Conference on Machine Learning, ICML 2007, pp. 273–280. ACM Press, New York (2007)
11. Joosten, B.: Creating a Havannah Playing Agent. Bachelor's thesis, Maastricht University, Maastricht, The Netherlands (2009)
12. Knuth, D.E., Moore, R.W.: An Analysis of Alpha-Beta Pruning. Artificial Intelligence 6(4), 293–326 (1975)

13. Kocsis, L., Szepesvári, C.: Bandit Based Monte-Carlo Planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
14. Laramée, F.D.: Using N-Gram Statistical Models to Predict Player Behavior. In: Rabin, S. (ed.) AI Game Programming Wisdom, pp. 596–601. Charles River Media, Hingham (2002)
15. Lee, C.-S., Wang, M.-H., Chaslot, G.M.J.-B., Hoock, J.-B., Rimmel, A., Teytaud, O., Tsai, S.-R., Hsu, S.-C., Hong, T.-P.: The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. IEEE Transactions on Computational Intelligence and AI in Games 1(1), 73–89 (2009)
16. Lorentz, R.J.: Improving Monte–Carlo Tree Search in Havannah. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) CG 2010. LNCS, vol. 6515, pp. 105–115. Springer, Heidelberg (2011)
17. Nijssen, J(P.) A.M., Winands, M.H.M.: Enhancements for Multi-Player Monte-Carlo Tree Search. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) CG 2010. LNCS, vol. 6515, pp. 238–249. Springer, Heidelberg (2011)
18. Rimmel, A., Teytaud, F.: Multiple Overlapping Tiles for Contextual Monte Carlo Tree Search. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Ekárt, A., Esparcia-Alcazar, A.I., Goh, C.-K., Merelo, J.J., Neri, F., Preuß, M., Togelius, J., Yannakakis, G.N. (eds.) EvoApplicatons 2010. LNCS, vol. 6024, pp. 201–210. Springer, Heidelberg (2010)
19. Rimmel, A., Teytaud, F., Teytaud, O.: Biasing Monte-Carlo Simulations through RAVE Values. In: van den Herik, H.J., Iida, H., Plaat, A. (eds.) CG 2010. LNCS, vol. 6515, pp. 59–68. Springer, Heidelberg (2011)
20. Shannon, C.E.: Predication and Entropy of Printed English. The Bell System Technical Journal 30(1), 50–64 (1951)
21. Stankiewicz, J.A.: Knowledge-based Monte-Carlo Tree Search in Havannah. Master's thesis, Maastricht University, Maastricht, The Netherlands (2011)
22. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
23. Teytaud, F., Teytaud, O.: Creating an Upper-Confidence-Tree Program for Havannah. In: van den Herik, H.J., Spronck, P. (eds.) ACG 2009. LNCS, vol. 6048, pp. 65–74. Springer, Heidelberg (2010)
24. Teytaud, F., Teytaud, O.: On the Huge Benefit of Decisive Moves in Monte-Carlo Tree Search Algorithms. In: Yannakakis, G.N., Togelius, J. (eds.) Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG 2010), pp. 359–364. IEEE Press (2010)
25. Winands, M.H.M., Björnsson, Y.: $\alpha\beta$-based Play-outs in Monte-Carlo Tree Search. In: 2011 IEEE Conference on Computational Intelligence and Games (CIG 2011), pp. 110–117. IEEE Press (2011)