Creating an Upper-Confidence-Tree program for Havannah

F. Teytaud and O. Teytaud

TAO (Inria), LRI, UMR 8623(CNRS - Univ. Paris-Sud),
bat 490 Univ. Paris-Sud 91405 Orsay, France, fteytaud@lri.fr

Abstract. Monte-Carlo Tree Search and Upper Confidence Bounds provided huge improvements in computer-Go. In this paper, we test the generality of the approach by experimenting on another game, Havannah, which is known for being especially difficult for computers. We show that the same results hold, with slight differences related to the absence of clearly known patterns for the game of Havannah, in spite of the fact that Havannah is more related to connection games like Hex than to territory games like Go.

1 Introduction

This introduction is based on [21]. Havannah is a 2-players board game (black vs white) invented by Christian Freeling [21, 18]. It is played on an hexagonal board of hexagonal locations, with variable size (10 hexes per side usually for strong players).

White starts, after which moves alternate. The rules are simple:

- Each player places one stone on one empty cell. If there's no empty cell and if no player has won yet, the game is a draw (very rare cases).
- A player wins if he realizes:
  - a ring, i.e. a loop around one or more cells (empty or not, occupied by black or white stones);
  - a bridge, i.e. a continuous string of stones from one of the six corner cells to another of the six corner cells;
  - a fork, i.e. a connection between three edges of the board (corner points are not edges).

These figures are presented on Fig. 1 for the sake of clarify.

Although computers can play some abstract strategy games better than any human, the best Havannah-playing software plays weakly compared to human experts. In 2002 Freeling offered a prize of 1000 euros, available through 2012, for any computer program that could beat him in even one game of a ten-game match.

Havannah is somewhat related to Go and Hex. It's fully observable, involves connections; also, rings are somewhat related to the concept of eye or capture in the game of Go. Go has been for a while the main target for AI in games, as it's

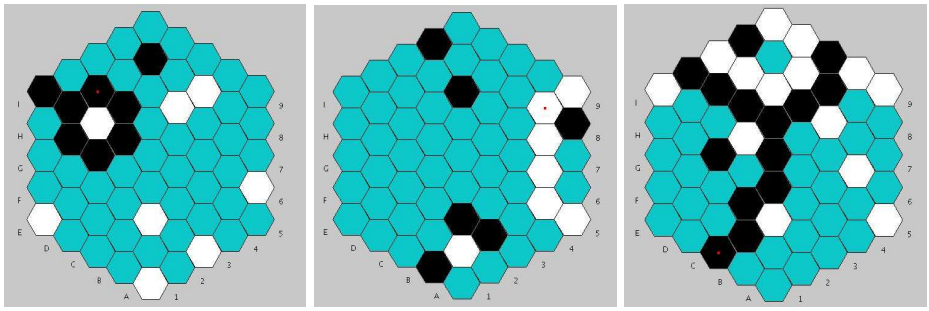


Fig. 1. Three finished games: a ring (a loop, by black), a bridge (linking two corners, by white) and a fork (linking three edges, by black).

a very famous game, very important in many Asian countries; however, it is no longer true that computers are only at the level of a novice. Since MCTS/UCT (Monte-Carlo Tree Search, Upper Confidence Trees) approaches have been defined [6, 9, 14], several improvements have appeared like First-Play Urgency [20], Rave-values [4, 12], patterns and progressive widening [10, 7], better than UCB-like (Upper Confidence Bounds) exploration terms [16], large-scale parallelization [11, 8, 5, 13], automatic building of huge opening books [2]. Thanks to all these improvements, MoGo has already won games against a professional player in 9x9 (Amsterdam, 2007; Paris, 2008; Taiwan 2009), and recently won with handicap 6 against a professional player, Li-Chen Chien (Tainan, 2009), and with handicap 7 against a top professional player, Zhou Junxun, winner of the LG-Cup 2007 (Tainan, 2009). The following features make Havannah very difficult for computers, perhaps yet more difficult than the game of Go:

- few local patterns are known for Havannah;
- no natural evaluation function;
- no pruning rule for reducing the number of reasonable moves;
- large action space (271 for the first move with board size 10).

The advantage of Havannah, as well as in Go (except in pathological cases for the game of Go), is that simulations have a bounded length: the size of the board.

The goal of this paper is to investigate the generality of this approach by testing it in Havannah.

By way of notation, $x \pm y$ means a result with average x, and 95% confidence interval $[x - y, x + y]$ (*i.e.* y is two standard deviations).

2 UCT

Upper Confidence Trees are the most straightforward choice when implementing a Monte-Carlo Tree Search. The basic principle is as follows. As long as there is time before playing, the algorithm performs random simulations from a UCT tree leaf. These simulations (playouts) are complete possible games, from the

root of the tree until the game is over, played by a random player playing both black and white. In its most simple version, the random player, which is used when in a situation s which is not in the tree, just plays randomly and uniformly among legal moves in s; we did not implement anything more sophisticated, as we are more interested in algorithms than in heuristic tricks. When the random player is in a situation s already in the UCT tree, then its choices depend on the statistics: number of wins and number of losses in previous games, for each legal move in s. The detailed formula used for choosing a move depending on statistics is termed a bandit formula, discussed below. After each simulation, the first situation of the simulated game that was not yet in memory is archived in memory, and all statistics of numbers of wins and numbers of defeats are updated in each situation in memory which was traversed by the simulation.

Bandit formula

The most classical bandit formula is UCB (Upper Confidence Bounds [15, 3]); Monte-Carlo Tree Search based on UCB is termed UCT. The idea is to compute a exploration/exploitation score for each legal move in a situation s for choosing which of these moves must be simulated. Then, the move with maximal exploration/exploitation score is chosen. The exploration/exploitation score of a move d is typically the sum of the empirical success rate $score(d)$ of the move d, and of an exploration term which is strong for less explored moves. UCB uses Hoeffding's bound for defining the exploration term:

$$exploration_{Hoeffding} = \sqrt{\log(2/\delta)/n} \tag{1}$$

where n is the number of simulations for move d. δ is usually chosen linear as a function of the number m of simulations of the situation s: δ is linear in m. In our implementation, with a uniform random player, the formula was empirically set to:

$$exploration_{Hoeffding} = \sqrt{0.25\log(2+m)/n}. \tag{2}$$

[1, 17] has shown the efficiency of using Bernstein's bound instead of Hoeffding's bound, in some settings. The exploration term is then:

$$exploration_{Bernstein} = \sqrt{score(d)(1-score(d))2\log(3/\delta)/n} + 3\log(3/\delta)/n \tag{3}$$

This term is smaller for moves with small variance ($score(d)$ small or large).

After the empirical tuning of Hoeffding's equation 2, we tested Bernstein's formula as follows (with $p = score(d)$ for short)

$$exploration_{Bernstein} = \sqrt{4Kp(1-p)\log(2+m)/n} + 3\sqrt{2}K\log(2+m)/n. \tag{4}$$

Within the "2+", which is here for avoiding special cases for 0, this is Bernstein's formula for δ linear as a function of m.

We tested several values of K. The first value 0.25 corresponds to Hoeffding's bound except that the second term is added:

K	Score against Hoeffding's formula 2
0.250	0.503 ± 0.011
0.100	0.578 ± 0.010
0.030	0.646 ± 0.005
0.010	0.652 ± 0.006
0.001	0.582 ± 0.012
0.000	0.439 ± 0.015

Experiments are performed with 1000 simulations per move, with size 5. We tried to experiment values below 0.01 for K but with poor results.

Scaling of UCT

Usually, UCT provides better and better results when the number of simulations per move is increased. Typically, the winning rate with $2z$ simulations against UCT with z simulations is roughly 63% in the game of Go (see e.g. [11]). In the case of Havannah, with uniform random player (i.e. the random player plays uniformly among legal moves) and bandit formula as in Eq. 2, we get the following results for the UCT tuned as above ($K = 0.25$):

Number of simulations of both player	Success rate
250 vs 125	0.75 ± 0.02
500 vs 250	0.84 ± 0.02
1000 vs 500	0.72 ± 0.03
2000 vs 1000	0.75 ± 0.02
4000 vs 2000	0.74 ± 0.05

These experiments are performed on a board of size 8.

3 Guiding exploration

UCT-like algorithms are quite strong for balancing exploration and exploitation. On the other hand, they provide no information for unexplored moves, and on how to choose among these moves; and little information for loosely explored moves. Various tricks have been proposed around this: First Play Urgency, Progressive widening, Rapid Action Value Estimates (RAVE).

3.1 Progressive widening/unpruning

In progressive widening [10, 7, 19], we first rank the legal moves at a situation s according to some heuristic: the moves are then renamed $1, 2, \ldots, n$, with $i < j$ if move i is preferred to move j for the heuristic. Then, at the m^{th} simulation of a node, all moves with index larger than $f(m)$ have $-\infty$ score (i.e. are discarded), with $f(m)$ some non-decreasing mapping from $\mathbb{N}$ to $\mathbb{N}$. It was shown in [19] that this can work even with random ranking, with $f(m) = \lfloor Km^{1/4} \rfloor$ for some constant K. In [10] it was shown that $f(m) = \lfloor Km^{1/3.4} \rfloor$ for some constant K

performs well in the case of Go, with a pattern-based heuristic. The algorithm proposed in [7] and now used also in MoGo is a bit different: an exploration term depending on a pattern-based heuristic and decreasing logarithmically with the number of simulations of this move is added to the score; for move d in situation s,

$$newScore(d,s) = score(d,s) + H(d,s)/\log(2+m),$$

where as previously n is the number of simulations including move d at situation s. In the case of Havannah, we have no such heuristic. We decided to use heuristic-free progressive widening, as it was shown in [19] that an improvement can be provided even if no heuristic is available (i.e. the order is arbitrary). This idea of using progressive widening without heuristic is a bit counter-intuitive. However, consider the following case. Consider a node of a tree which is explored 50 times only (this certainly happens for many nodes deep in the tree). If there are 50 legal moves at this node, then the 50 simulations will be distributed on the 50 legal moves (one simulation for each legal move), if you use the standard UCB formula instead of (without) progressive widening. Meanwhile, progressive widening will sample a few moves only, e.g. 4 moves, and sample much more the best of these 4 moves - this is likely to be better than taking the average of all moves as an evaluation function. We experimented with 500 simulations per move, size 5, various constants P and Q for the progressive widening $f(m) = Q\lfloor m^P \rfloor$:

Q, P	Success rate against no prog. widening
1, 0.7	0,496986 ± 0,014942
1, 0.8	0.51 ± 0,0235833
1, 0.9	0.50 ± 0,0145172
4, 0.4	0,500454 ± 0,0134803
4, 0.7	0.49 ± 0,0181818
4, 0.9	0,485101 ± 0,0172619

These experiments were performed with the exploration formula given in Eq. 2. We tested various other parameters for Q and P, without seeing a significant improvement. Perhaps improvements are possible by jointly tuning the Hoeffding's bound and the progressive widening formula.

3.2 Rapid Action Value Estimate

In the case of Go [4, 12] propose to average the score with a permutation-based statistical estimate. Precisely, the score for a move d in situation s simulated n times becomes:

$$newScore(d,s) = (1-\alpha(n))score(d,s) + \alpha(n)rave(d,s) + exploration$$

where

- $score(d,s)$ is the ratio of won simulations among simulations with situation s and move d in s, $rave(d,s)$ is the proportion of won games among games *containing* move d after situation s.

– $\alpha(n) \to 0$ as $n \to \infty$, whereas $\alpha(n)$ is close to 1 for n small, so that the heuristic *rave* values are used initially, and eventually they are replaced by "real" values.

The difference between $score(d, s)$ and $rave(d, s)$ is that the proportion of won games in $rave()$ is computed among all simulations *containing d as move after situation s* and not only all simulations *with d as move in situation s*. In the game of Go, RAVE values are a great improvement. However, they involve complicated implementations due to captures and re-captures. In the case of Havannah there's no such problem, and we'll see that the results are good. We used the following formula:

$$\alpha(n) = R/(R+n), \quad exploration = exploration_{Hoeffding} = \sqrt{K \log(2+m)/n}.$$

R was empirically set to 50 (on games with size 5 with 1000 simulations per move); intuitively, R is the number of simulations before the weight of "RAVE" values is equal to the weight of "greedy" values. All scores below are against UCT with Eq. 2.

size 5	R=50, K=0.25, size 5, 1000 simulations/move	47.26% ± 4.0 %
	R=50, K=0.05, size 5, 1000 simulations/move	60.46% ± 2.9 %
	R=50, K=0, size 5, 1000 simulations/move	95.33% ± 0.01 %
size 8	R=50, K=0, size 8, 1000 simulations/move	100% on 1347 runs

$K = 0$ is then the best constant. This was also pointed out in [16] for the game of Go. We then tested larger numbers of simulations, i.e. 30 000 simulations per move. Disappointingly but consistently with [16], we had to change the coefficients in order to get positive results, whereas the tuning of UCT is seemingly more independent of the number of simulations per move.

R=50, K=0, size 5, 30 000 simulations/move	0.53 ± 0.02
R=50, K=0.25, size 5, 30 000 simulations/move	0.47 ± 0.04
R=50, K=0.05, size 5, 30 000 simulations/move	0.60 ± 0.02
R=50, K=0.02, size 5, 30 000 simulations/move	0.60 ± 0.03
R=5, K=0.02, size 5, 30 000 simulations/move	0.61 ± 0.06
R=20, K=0.02, size 5, 30 000 simulations/move	0.66 ± 0.03

The first line corresponds to the configuration empirically chosen for 1000 simulations per move; its results are disappointing, almost equivalent to UCT, for these experiments with 30 000 simulations per move. The second line uses the same exploration constant as UCT, but it's seemingly too much. Then, in the following lines, using a weaker exploration and a small value of R, we get better results. [16] points out that, with big simulation times, $K = 0$ was better, but an exploration bonus depending on patterns was used instead.

As a conclusion, for large numbers of simulations, RAVE is not as efficient as for small values (when compared to UCT). Perhaps better tuning could yield better results. The main weaknesses of RAVE are:

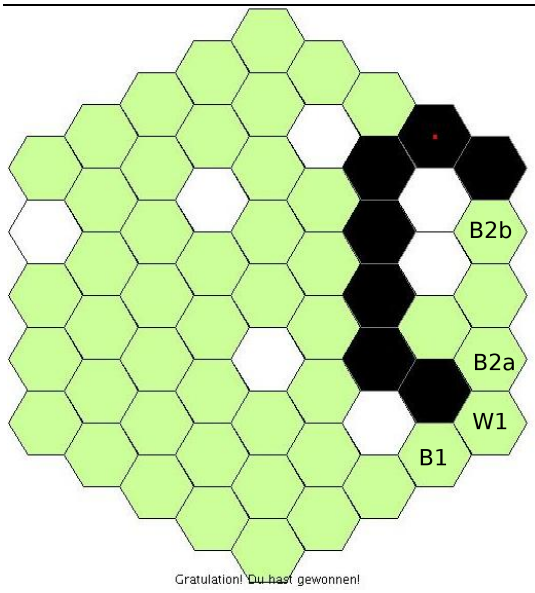
– tuning is required for each new number of simulations;

– the results for large numbers of simulations are less impressive (yet significant).

On the other hand, the efficiency increases with the size of the action space - this is promising for the application of RAVE to large action spaces.

4 Games against Havannah-Applet

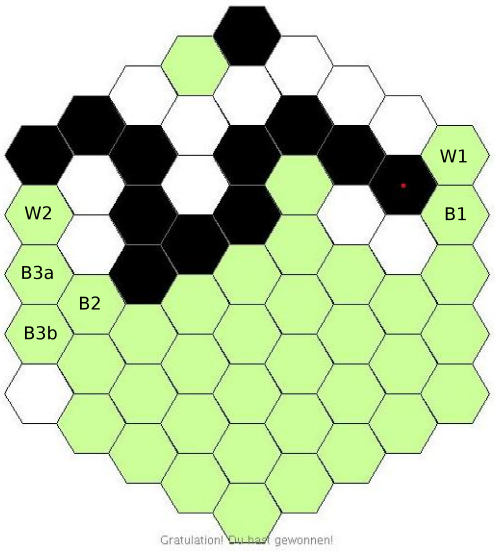
We tested our program against Havannah-Applet `http://dfa.imn.htwk-leipzig.de/havannah/`, recommended by the MindSports association as the only publicly available program that plays by the Havannah rules. There are 30 seconds per move, but we restricted our program was running in 8 secondes per move (first game) and 2.5 seconds per move (second game). In both cases, our program, based on RAVE, no exploration term, no progressive widening, was black (white starts and has therefore the advantage in Havannah). The first game (played with 8 seconds per move for our program, against 30s for the opponent) is presented in Fig. 2. Consistently, the estimated success



Success rate estimated by Havannah-Applet	Success rate estimated by our software
62.6%	46.4%
60.8%	47.2%
64.0%	51.2%
56.5%	54.8%
62.5%	65.4%
57.3%	63.7%
71.0%	88.8%
0.5%	

Fig. 2. Left: the result of the game played against Havannah-Applet in size 5. Our program won very quickly, by a nice multiple constraint: the white opponent must play *W1* (unless black wins by bridge). Then, black can connect to the lower-right edge with *B1*. Then, black has two opportunities for connecting to the right edge, *B2a* and *B2b*; white can only block one and black then realizes a fork. As seen on the estimated success rate, Havannah-Applet did not suspect this attack before the very last move. Right: estimated success rate for each of the opponents.

rate is lower than 50 % at the beginning (as the opponent, playing first, has the advantage initially). It then increases regularly until the end. The second game is presented in Fig. 3, with only 7000 simulations (nearly 2 seconds) per move. Consistently again, the estimated success rate is lower than 50 % at the



Success rate estimated by Havannah-Applet	Success rate estimated by our software
62.3%	45.7%
60.3%	48.2%
50.6%	49.9%
43.0%	55.9%
40.4%	52.6%
52.6%	55.4%
40.7%	56.2%
42.0%	63.5%
31.1%	60.0%
42.2%	75.8%
36.8%	68.6%
27.4%	78.1%
3.7%	

Fig. 3. Left: the result of the game played against Havannah-Applet in size 5. Our program was playing black and won by resignation. White has to play *W1*, otherwise black realises a bridge. Then, Black plays *B1* and is connected to a second side. Next, White must play *W2* (if Black plays *W2* then Black has a fork). Finally, Black can play *B2* and with with move, is connected to the third side (by *B3a* or *B3b* (White can't avoid this connection). Right: estimated success rate for each of the opponents.

beginning (as the opponent, playing first, has the advantage initially). It then increases regularly until the end.

5 Discussion

We could clearly validate in the case of Havannah the efficiency of some well known techniques coming from computer-Go, showing the generality of the MCTS approach. Essentially:

- The efficiency of Bernstein's formula, in front of Hoeffding's formula, is clear (up to 65%).
- The constant success rate of UCT with $2k$ simulations per move, against UCT with k simulations per move, nearly holds in the case of Havannah (nearly 75%, whereas it is usually around 63% for MCTS in the game of Go

[11]). However, the success rate is higher than in the case of the game of Go (around 75%).
- The efficiency of the RAVE heuristic is clearly validated. The main strength is that the efficiency increases with the size, reaching 100 % on 1347 games in size 8. On ther other hand, RAVE becomes less efficient, and requires tuning, when the number of simulations per move increases - however, we keep 66% of success rate).
- Progressive widening, in spite of the fact that it was shown in [19] that it works even without heuristic, was not significant for us. In the case of Go, progressive widening was shown very efficient in implementations based on patterns [10, 7].
- Our program could defeat Havannah-Applet easily, whereas it was playing as black, and with only 2s per move instead of 30s (running on a single core). Running more experiments was difficult due to lack of automated interface.

References

1. J.-Y. Audibert, R. Munos, and C. Szepesvari. Use of variance estimation in the multi-armed bandit problem. In *NIPS 2006 Workshop on On-line Trading of Exploration and Exploitation*, 2006.
2. P. Audouard, G. Chaslot, J.-B. Hoock, J. Perez, A. Rimmel, and O. Teytaud. Grid coevolution for adaptive simulations; application to the building of opening books in the game of go. In *Proceedings of EvoGames*, 2009.
3. P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2/3):235–256, 2002.
4. B. Bruegmann. Monte carlo go. *Unpublished*, 1993.
5. T. Cazenave and N. Jouandeau. On the parallelization of UCT. In *Proceedings of CGW07*, pages 93–101, 2007.
6. G. Chaslot, J.-T. Saito, B. Bouzy, J. W. H. M. Uiterwijk, and H. J. van den Herik. Monte-Carlo Strategies for Computer Go. In P.-Y. Schobbens, W. Vanhoof, and G. Schwanen, editors, *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91, 2006.
7. G. Chaslot, M. Winands, J. Uiterwijk, H. van den Herik, and B. Bouzy. Progressive strategies for monte-carlo tree search. In P. Wang et al., editors, *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pages 655–661. World Scientific Publishing Co. Pte. Ltd., 2007.
8. G. Chaslot, M. Winands, and H. van den Herik. Parallel Monte-Carlo Tree Search. In *Proceedings of the Conference on Computers and Games 2008 (CG 2008)*, 2008.
9. R. Coulom. Efficient selectivity and backup operators in monte-carlo tree search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, 2006.
10. R. Coulom. Computing elo ratings of move patterns in the game of go. In *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.
11. S. Gelly, J. B. Hoock, A. Rimmel, O. Teytaud, and Y. Kalemkarian. The parallelization of monte-carlo planning. In *Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO 2008)*, pages 198–203, 2008. To appear.

12. S. Gelly and D. Silver. Combining online and offline knowledge in uct. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM Press.
13. H. Kato and I. Takeuchi. Parallel monte-carlo tree search with simulation servers. In *13th Game Programming Workshop (GPW-08)*, November 2008.
14. L. Kocsis and C. Szepesvari. Bandit-based monte-carlo planning. In *ECML'06*, pages 282–293, 2006.
15. T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.
16. C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong. The computational intelligence of mogo revealed in taiwan's computer go tournaments. *IEEE Transactions on Computational Intelligence and AI in Games*, 2009 (accepted).
17. V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical Bernstein stopping. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 672–679, New York, NY, USA, 2008. ACM.
18. R. W. Schmittberger. *New Rules for Classic Games*. Wiley, 1992.
19. Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, volume 21, 2008.
20. Y. Wang and S. Gelly. Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii*, pages 175–182, 2007.
21. Wikipedia. Havannah, 2009.