Improving Monte–Carlo in Havannah

Richard J. Lorentz

Department of Computer Science
California State University
Northridge CA 91330-8281 USA
lorentz@csun.edu

Abstract. Havannah is a game played on an hexagonal board of hexagons where the base of the board typically has between from 4 and 10 hexagons. The game is known to be difficult to program. We study an MCTS based approach to programming Havannah using our program named Wanderer. We experiment with various aspects and enhancements of the basic MCTS algorithms and demonstrate that at normal time controls of approximately 30 seconds per move Wanderer can make quite strong moves with bases of size 4 or 5, and play a reasonable game with bases of size 6 or 7. At longer time controls (10 minutes per move) Wanderer appears to play nearly perfectly with base 4, is difficult for most humans to beat at base 5, and gives a good game at bases 6 and 7.

Keywords: Havannah, Monte-Carlo algorithm, MCTS.

1 Introduction

Havannah is a fairly new game, invented in 1979 by Christian Freeling. It is a two-player game played on an hexagonal board of hexagons where a side of the hexagonal board typically has from four to ten hexagons. The number of hexagons along a side is usually referred to as the *base*.

The two players (White and Black) alternate playing on empty cells. White plays first. The first player to form a ring, a fork, or a bridge wins the game. A ring is a connected set of cells of the same color that surrounds at least one other cell. The surrounded cell(s) may be empty or occupied by either color. A fork is a connected set of cells that touches at least three different edges. A bridge is a connected set of cells that touches at least two different corners. A corner cell is not considered to be an edge. If the board is full and no winning structure has been formed, the game is deemed a draw. In practice this rarely happens. In the thousands of self-play games we have played with our program, Wanderer, we have yet to see a draw.

Fig. 1 shows an example Havannah position where the board has base 6. It does not show an actual game but simply illustrates the various winning positions. White can win by forming a bridge by playing either F10 or G11. White can form a winning loop by playing C6 and similarly Black can form a loop by playing H5. If Black plays A2 he forms a winning fork. Black can achieve a different fork by playing either G5, F4, or I6.

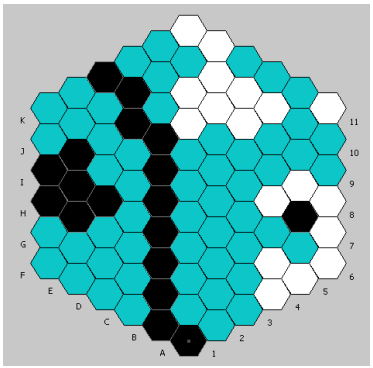

Fig. 1. A base 6 Havannah position

We have written a program to play Havannah which we call Wanderer. It is with this program that we perform all of our tests, which forms the basis for most of our algorithmic discussions, and which will be described in more detail below. For now suffice it to say that it is uses a Monte-Carlo Tree Search (MCTS) based approach and so evaluations returned are expressed in percentages.

Playing a good game of Havannah requires both strategic and tactical skills. The larger the base, the more strategy required. Base 4 Havannah is almost purely a tactical game while base 10 Havannah requires considerable strategy to play well. Not surprisingly, the larger the base the worse computer programs are able to play. Of course one of the main reasons for this is the much larger state space on large boards.

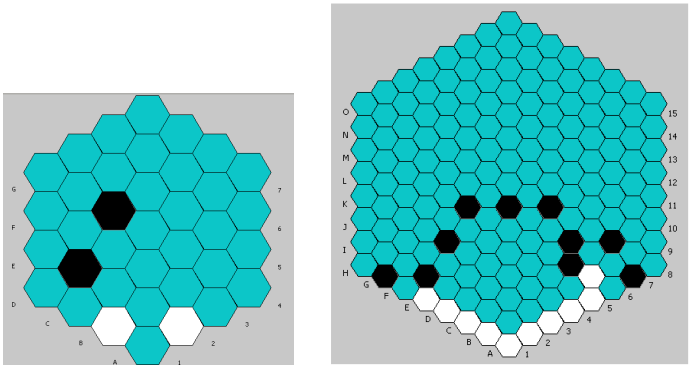

Fig. 2. A comparison of a program's capabilities on small versus large board sizes

Consider Fig. 2. On the base 4 board on the left Wanderer is able to determine that White has a forced win (by playing B4) in about 45 minutes while in the position on the right after one hour's calculation Wanderer feels White's position is better, returning an evaluation of 68% for the move C3. The truth of the matter in the second position is that Black has an easily won game. The bottom two edges cannot be

prevented from being connected (note the chain of unbreakable joints from one edge to the other) and then it is a simple matter to connect to a third edge. Wanderer is oblivious to this because: (1) it does not see the unbreakable nature of the connection and (2) there is enough space on the bottom of the board that it feels it has a good chance of forming a loop.

In what follows we will briefly review the basic MCTS approach to game programming that we used, especially as it applies to Havannah and explain the kind of testing we did. We then discuss some known enhancements to MCTS and explain how they, or novel modifications of them, can be applied to improve a Havannah program. We follow with a slightly more detailed discussion of the experimental results and conclude with some final remarks.

2 Wanderer, an Havannah Playing Program Test Bed

Wanderer is a Monte-Carlo Tree Search (MCTS) based program and despite Hartmann's admonition [13] we will assume that the reader is familiar with the basic ideas of Monte-Carlo search, random playouts, and MCTS. Wanderer is constructed to use the UCT approach [14] to exploration/exploitation and it is from this point that our research commences. We performed our experimental testing in two different ways. Most of the tests were self tests with one version of Wanderer playing against another version with time controls set at "normal tournament rates", namely 30 seconds per move. Since Wanderer is still in the early stages of development it often became obvious from the observed quality of moves which version was superior. In any event, we would run a maximum of 100 test games where the first two moves of the game were chosen at random and the remainder of the moves were selected by the engine. We evolved our algorithms by first working on boards with base 4 and then incrementally increasing the board size while exploring and refining algorithms. The results presented here were developed with all bases in mind, but virtually all of the test cases were performed on boards with base 6.

Another kind of testing was done by entering Wanderer in a turn based game playing site called "Little Golem" (LG) [18] where opponents have on average a maximum of 48 hours to make a move (though most participants make Havannah moves more quickly). For these games Wanderer is set to move after 10 minutes of calculation. Since LG does not distinguish among board sizes in its rating data we need to pay more attention to the game results at different sizes rather than the rating calculated by LG.

As in Teytaud and Teytaud [16] we tried two different UCT exploration terms which they referred to as Bernstein and Hoeffding, but contrary to their results we found that the more usual Hoeffding term was to be preferred. This may be because our tests were performed with significantly longer thinking time than was done by Teytaud and Teytaud.

One other point concerning the random playouts needs to be addressed. The authors of two other Havannah programs, Richard Pijl and Thomas Reinhardt, have informally offered information about the workings of their programs. They quite reasonably suggest that the information gained by a random playout becomes less

useful as the playout gets deeper into the game. Thus they both cutoff the random playout after some number of moves rather than letting it continue to the end of the game. Our experience contradicts this natural idea and we find that Wanderer performs better if random playouts are allowed to continue until the end. However, we do find that some modification of the playouts is useful as they near the end of the game. This will be discussed in the section on smart playouts.

3 Improving the Basic MCTS Algorithm

MCTS algorithms are still rather new and not all that well understood. Nevertheless a number of techniques are emerging that improve the basic algorithms. We discuss five techniques that when combined significantly improve the playing strength of Wanderer.

3.1 Improving the Random Playouts

It is surprising that a strong computer player can be constructed with no evaluation function, simply relying on information gained from random playouts. But the mysterious fact remains that large numbers of random games from a given position can give a good estimate of the status of that position. By extension, one would expect that by having the playouts make moves that are better than random one could gain more accurate information about a position and with fewer playouts. There is some truth to this. For example, in the game of Go guiding the random playouts through the use of patterns is known to be a useful technique [12]. However, producing useful smarter playouts (sometimes referred to as *heavy playouts*) is not trivial. Often the time required to find smarter moves reduces the number of random playouts that can be made in a given amount of time to such an extent that any benefit is lost. Other times even though the moves are better and the time penalty is small or nonexistent it still does not improve the quality of moves actually being made by the program. Currently, discovering effective ways to make playouts smarter has a disturbingly large trial and error factor.

We use two main techniques to make the playouts smarter. The first was suggested to us at the 14th Computer Olympiad, held in Pamplona in May, 2009 [7]. If during a random playout a player can make a move and achieve a winning configuration (a "mate in one") then that move should always be made. Unfortunately, detecting winning positions is a time consuming process in Havannah, especially checking for loops. We are able to get approximately four times more random playouts without the check than with it. But the improvement in the program with the check is significant and so must be used.

We also employ two simple extensions to this idea. The first is to check not only if the player has a wining move but also if the opponent does. If the opponent has a "mate in one" then the playout will play on that point in an attempt to prevent the immediate loss. The second extension is an attempt to improve speed while using the technique. Though we claimed earlier that it is advantageous to continue random playouts until the end of the game, one move wins become less important as the game

proceeds and since checking for them is time consuming we only check for these moves early in the playout. With some experimental tuning we found that if we check for winning moves in the playout for the first 5 * base moves of the random playout (base is the base size of the board) we get the advantages of checking for winning moves with an approximate 20% execution speedup.

The second technique for improving the playouts has to do with trying to find moves that are near where the action is. After much experimentation we conclude that during a playout we should encourage moves to be made on hexagons that are adjacent to hexagons that have the same color (one hexagon away) or are two hexagons away from a hexagon of the same color. The time penalty for this check is small, less than 5%, and again the improvement in play is clearly apparent.

Combining these two techniques makes for a clearly stronger player that wins more than two thirds of the games against the same program without these improvements.

Other attempts to improve playouts have yielded disappointing results. For example, in most situations making a move that is adjacent to two other *adjacent* hexagons of the same color is a bad move. However, preventing such moves in the smart playouts had no effect on the overall strength of the moves being made even though the cost in time was unnoticeable. Similarly, most moves made tend to be near the last move made by the same player. This is often referred to as the proximity heuristic [9]. Despite our preconceived "certainty" that this can be used to advantage in Havannah, we have yet to obtain any positive results. Finally, concerning the moves that are two hexagons away, we tested a modification where we only considered such moves when the two hexagons between the proposed move and the existing hexagon of the same color are vacant. The idea is to create "joints" as seen in the second board of Fig. 2. This appears to provide a minor improvement to Wanderer, but only from an intuitive standpoint. The experimental results are inconclusive.

3.2 Recognizing Forced Wins, Losses, and Draws

Basic MCTS algorithms do not directly deal with positions that are forced wins, losses, or draws. For example, even in a position with a forced win MCTS will continue making random playouts, moving towards the optimal variation and the winning percentage will grow as it pushes towards 100%. There is a natural way to deal with forced wins and, symmetrically, forced losses with minor modifications to the MCTS tree and using similar techniques we can also handle forced draws. Doing this provides three advantages. First, we are able to actually prove the status of a position rather than assume the fact with high probability. Second, once the value of a root position has been proved the move can be made, thus saving time. This is particularly important with small Havannah boards. Finally, if a node's status is determined we are able to prune all children of that node from the MCTS tree. This saving of memory allows us to run the program for much longer times than would otherwise be possible. We would not have been able to find the winning move in the first board in Fig. 2 without using these techniques.

The basic idea is to view the MCTS tree as also being a kind of mini-max tree. We add status values to MCTS tree nodes where the possible values are WIN, LOSS, DRAW, and NONE and when the status value of a node changes we propagate status values up the tree in a mini-max type fashion.

By default nodes have the value NONE. If a node is added to the tree that ends the game (a winning configuration has been obtained) mark the node as WIN meaning that the player who just made that move won. Propagation is accomplished in the obvious way: if any child of a node has status WIN then that node should be set to LOSS. If every child of a node has status LOSS then set that node to WIN. If the status of a node is set, prune all child nodes of that node and free the associated memory. If the status of the root is set then the tree value has been determined and we are able to make the appropriate move.

Similarly for draws, if a node corresponds to a position where there are no remaining moves it is flagged as DRAW. For propagation, if every child of a node is either DRAW or LOSS then that node can be flagged as a DRAW. And as before, if the root is flagged as DRAW then we make a move corresponding to one of the children that is also a DRAW.

3.3 Initializing Total Simulation and Win Counts

When creating new nodes for the MCTS tree the win counts and the total playout counts are usually initialized to zero. However, if there is prior knowledge of the quality of the move represented by a node relative to its siblings, then it sometimes makes sense to initialize these values to something other than zero that reflects the perceived relative value of that position [10]. For example, initializing both the simulation counts and the win counts to high values indicates that the position is likely to be favorable (thus the high initial win count) and there is high confidence that this judgment is correct (because of the high initial playout count).

One of the reasons that Havannah is such a difficult programming challenge is that it is difficult to judge the relative merits of various candidate moves from a given position and hence we are given little guidance as to how we might initialize these node counts. Nevertheless, similar to the situation described in Section 3.1, there are some properties of moves that suggest superiority over other move choices. One such is the joint move, that is, a move that is two hexagons away from a hexagon of the same color and the two hexagons in between are vacant. Another is moving adjacent to a hexagon of the same color. On average these kinds of moves tend to be better than other moves.

We use these two ideas to initialize the counts in new MCTS nodes. Specifically, in all cases we set the total playout count to 40 and in the case of a joint we set the win count to 30, for a neighboring move we set the win count to 40, and in all other cases we set the win count to 5. These values were selected after a short amount of trial and error and can probably be improved with further tuning. Nevertheless, this simple idea markedly improves the playing strength of Wanderer.

3.4 Progressive Widening

It has been shown that it is possible to give MCTS a little assistance by first considering only the "best" children of a node and then gradually adding the other children until at some point all children are actually being examined [3, 4, 6]. Ordinarily this is done by ordering the child moves using an evaluation function, initially putting the highest evaluating moves in the MCTS tree, and then gradually adding moves with lower evaluation values as the node gets visited more and more often. As explained in [16] since no reasonable evaluation functions are known for Havannah we must resort to "heuristic-free progressive widening" which basically amounts to using an evaluation function that produces random values for each position. Though counter intuitive, it seems this technique can provide benefit. We propose a modification that, in the case of Havannah, performs better that the purely random version.

We attempt to recover an evaluation from the results provided by the MCTS search. At each node we allow all possible moves in the early stages. Then after that node has been visited a certain number of times (experimentally we have had reasonable success with a value of 50,000 visits) we use the MCTS win counts as an evaluation. With this as an evaluation function we then proceed to use standard progressive widening techniques. Again experimentally we find that slowly adding back all the MCTS nodes during the next 500,000 visits works well. On boards with bases 4 and 5 progressive widening seems to be even more useful than on boards with base 6, but the first move advantage on smaller boards tends to obscure this advantage.

3.5 The Killer RAVE Heuristic

The Rapid Action Value Estimate (RAVE) [10] tries to deal with the same problem that progressive widening deals with. It is another attempt to select good moves from a position before a significant number of random playouts have been performed from that position. In the case of RAVE the assumption is that good moves made later in the game would probably have also been good moves had they been made earlier. In the case of Go this is a reasonable assumption and RAVE has had considerable success there. This is not so clearly the case with Havannah.

Our experiments have shown that in the case of Havannah the basic RAVE approach offers no advantage. This is not too surprising since two different moves can easily completely change the course, and therefore the subsequent moves, of a game. Nevertheless, a restriction of RAVE to only certain moves appears to offer significant promise. The idea is to select only the most important moves that appear in the game and just use them for computing RAVE values. This reminds us of the killer heuristic found in mini-max programming [1] since in both cases we are exploiting moves that have been flagged as strong in other parts of the tree. For this reason we refer to it as "killer RAVE."

As described in Section 3.1 winning and losing moves are particularly significant, especially when those moves are not too distant from the current position. By emphasizing these moves in a RAVE like manner when a node still has few visits we

find an improvement in the overall strength of Wanderer. We suspect even greater improvement is possible if we can find a way to increase the size of the killer pool.

4. Results

The improvements described above make for a significantly stronger player which is confirmed by self-play tests. The results are shown in Fig. 3. Tests were done on boards with base 6 with the current version of Wanderer playing against a version handicapped as described in the first column. We play a total of 200 games in each case and since White has a distinct advantage by being able to move first we test both ways, that is, each version gets to move first in 100 of the games. Also, to avoid multiple occurrences of the same or similar games in the test data each game starts with a random move from each player. The numeric entries correspond to the winning percentage of the full strength Wanderer against the other, handicapped version.

Wanderer Opponent	Wanderer as white	Wanderer as black
Pure MCTS/UCT with no enhancements	84 %	73 %
Playouts not enhanced with locality check	70 %	54 %
Playouts do not check for mate in 1	71 %	50 %
No Killer RAVE heuristic	70 %	41 %
No progressive widening	70 %	44 %
No playout count initialization	73 %	59 %

Fig. 3. Summary of Test Results.

A few observations are in order. The data clearly shows the advantage of moving first on a base 6 board. (For smaller boards, this is even more pronounced.) Indeed in every row White wins more games than Black. Running Wanderer against itself we find that White wins 59% of the games, providing further verification of the advantage of moving first and also proving a baseline for comparing the table entries.

The first row of the table clearly shows the advantage of using all the new techniques. Subsequent rows attempt to isolate the relative advantage each individual technique provides. Surprisingly, each of these rows shows fairly similar results when Wanderer is moving first but more variation when moving second.

We are tempted to rate the enhancements as follows. Most important is to initialize the playout count. This suggests that additional effort spent here tuning initialization values and finding other move properties to determine initialization values is likely to improve the program further. The two playout enhancements we put in a second category of techniques that we consider clearly important and likely to provide further improvements to Wanderer. We place progressive widening and the killer RAVE heuristic in a third category of techniques that appear promising but require more work to fully prove themselves worthy of inclusion.

We also tested Wanderer with longer running times by allowing it 10 minutes per move against various opponents on the turn based playing site LG. See Section 2. Though our research has mainly focused on improving Wanderer with base 6 we allow it to play games with bases of sizes 4 through 8, where only recently have we added base 8. Wanderer has done quite well and seems to have settled into a rating in

the mid 1800's putting it in the top 15 of about 400 players. However, even at this relatively slow move speed, Wanderer displays very little sense of strategy and this can be seen by its results when viewed according to the base size. This is summarized in Fig. 4 where we show the results of the most recent 145 games played on LG at the time of this writing.

base size	wins	losses
4	15	0
5	23	1
6	32	23
7	17	21
8	5	8

Fig. 4. Wanderer's results on the Little Golem turn based game playing site shown according to the base size of the board.

Looking at actual games played by Wanderer on LG we see a very noticeable improvement versus moves made at 30 seconds per move. Indeed we see that very often it is not until three or four minutes into its calculations before it finds a reasonable move.

5. Remarks

Wanderer made its first appearance in May 2009 at the 14th Computer Olympiad, held in Pamplona. This version of Wanderer used a pure MCTS/UCT approach with none of the enhancements described here. With just a few minor exceptions it corresponds to the version shown on the first line of Fig. 3. The only other competitor at the Olympiad was Shakti [17], a very similar program. The competition was on boards of size 5 and 8 and Wanderer outperformed Shakti only because it was able to do random playouts faster than Shakti. Since boards with base 5 provide more tactical games Wanderer was clearly superior. But with base 8 the difference was not so clear, even though Wanderer still won three out of four games. Also, from a human's point of view the quality of those games was quite low. The point of this research was to take the next step.

We have demonstrated that variations of techniques now fairly common in computer Go can significantly improve a Havannah program. At speeds of 30 seconds per move Wanderer can play a reasonable looking game with board bases from four to six. With higher bases the results are still rather disappointing. However, with ten minutes per move we have shown that we can give the average human a good game up to base seven, even if the human is given considerably more time to think.

Because of the first move advantage in Havannah it is common to employ a "swap rule." The swap rule attempts to remove this advantage by allowing Black to take White's first move as his own if he likes. Since LG utilizes the swap rule Wanderer deals with it in the most primitive way. On boards with base four, five, or six when White it always selects a random move and as Black it always swaps. On larger board

sizes it ignores the swap rule and always makes the move it considers best. More research needs to be done to properly take advantage of this rule.

It has become clear that as the quality of Havannah programs increases, the various bases need to be dealt with individually. For example, progressive widening has a greater impact with smaller bases. Another example is that the UCT constant needs to be adjusted according to the base. We found that the larger the base the smaller the UCT constant must be, meaning with larger board sizes we must exploit more at the expense of more careful exploration. This is because if the MCTS tree does not reach some critical depth it remains oblivious to any winning strategy that might be available.

Perhaps the most immediately compelling problem that needs to be addressed is finding a way for computer Havannah programs to deal with large board sizes and corresponding strategic concepts. Programs need to be able to understand positions like the second one in Fig. 2. Among other things this means that we need to find ways to more quickly focus on the more promising moves in a position.

References

1. Akl, S.G. and Newborn, M.M. (1977). The Principal Continuation and the Killer Heuristic. 1977 ACM Annual Conference Proceedings, pp. 466–473, ACM Press, New York, NY, USA.[KILLER]
2. Bruegmann, B.: Monte-Carlo go, unpublished manuscript (1993).
3. Cazenave, T.: Iterative widening. In 17th International Joint Conference on Artificial Intelligence (IJCAI-01) (2001) 523-528.
4. 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. In P. Wang et al., editors, Proceedings of the 10th Joint Conference on Information Sciences (2007) 655-661. [Progressive unpruing]
5. Coulom, R.: Efficient selectivity and back-up operators in Monte-Carlo tree search. In Proceedings Computers and Games 2006, Torino, Italy (2006).
6. Coulom, R: Computing Elo Ratings of Move Patterns in the Game of Go. In H. Jaap van den Herik, Jos W. H. M. Uiterwijk, Mark Winands and Maarten Schadd editors, Computer Games Workshop, Amsterdam, The Netherlands (2007) 113-124. [Progressive widening]
7. de Koning, J.: Personal communication.
8. Drake, P,: A. Pouliot, N. Schreiner, and B. Vanberg: The proximity heuristic and an opening book in Monte Carlo Go. Submitted, 2007.
9. Drake, P., Uurtamo, S.: Move Ordering vs Heavy Playouts: Where Should Heuristics Be Applied in Monte Carlo Go? In Proceedings of the 3rd North American Game-On Conference (2007).
10. Gelly, S. 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. [RAVE & PRIORS]
11. Gelly, S., Wang, Y.: Exploration exploitation in go: UCT for Monte-Carlo go. In Twentieth Annual Conference on Neural Information Processing Systems (2006).
12. Gelly, S., Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA, France, November 2006. [smarter playouts]
13. Hartmann, Dap, On the importance of self-contained papers, Journal of the International Computer Games Association, Vol 32 No. 4. 2009.

14. Kocsis, L., Szepesvari, C.: Bandit based Monte-Carlo planning, In 15th European Conference on Machine Learning (ECML) (2006) 282-293.
15. Mnih, V., C. Szepesv´ari, 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. [Bernstein UCT]
16. Teytaud, F., and Teytaud, O.: Creating an Upper-Confidence-Tree program for Havannah, In Proceedings Advances in Computer Games 12, to appear.
17. http://www.grappa.univ-lille3.fr/icga/program.php?id=600 [SHAKTI]
18. http://www.littlegolem.net/jsp/index.jsp