New Mathematics and Natural Computation
© World Scientific Publishing Company

HISTORY AND TERRITORY HEURISTICS FOR MONTE CARLO GO

BRUNO BOUZY

*Université Paris 5, UFR de mathématiques et d'informatique, C.R.I.P.5,
45, rue des Saints-Pères 75270 Paris Cedex 06 France,
bouzy@math-info.univ-paris5.fr*

Recently, the Monte Carlo approach has been applied to computer go with promising success. INDIGO uses such an approach which can be enhanced with specific heuristics. This paper assesses two heuristics within the 19x19 Monte Carlo go framework of INDIGO: the territory heuristic and the history heuristic, both in their internal and external versions. The external territory heuristic is more effective, leading to a 40-point improvement on 19x19 boards. The external history heuristic brings about a 10-point improvement. The internal territory heuristic yields a few points improvement, and the internal history heuristic has already been assessed on 9x9 boards in previous publications. Most of these heuristics were used by INDIGO at the 2004 Computer Olympiad.

Keywords: Monte Carlo go; Territory heuristic; History heuristic.

1. Introduction

Monte Carlo (MC) approaches to game playing are usual in hidden information games such as Poker or Scrabble, or in game containing randomness such as Backgammon. In Poker, POKI developed by the GAMES group at the University of Alberta uses simulations [2]. In Scrabble, MAVEN by Sheppard uses them [14,15] as well. Rollout experiments are also used at Backgammon [17]. The basic idea of MC game is due to Abramson [1]. To evaluate a position, the algorithm launches a given number of completely random games to the end and then scores them. The evaluation corresponds to the mean of the scores of those random games. Choosing a move in a position means playing each of the moves and maximize the evaluations of the positions obtained at depth 1. Several improvements can be made on the basic idea. The most common is progressive pruning [2]: when a candidate move has a sufficiently bad estimation compared to the best move estimation, it is pruned from the candidate list.

In go, GOBBLE by Brügmann [8], is the first 9x9 MC go program. It was written before 1993, and it used simulated annealing. In actual go games, the end of a game in the human meaning can be hard to detect. But in a random go game, the end of the game occurs when each intersection is either occupied by a stone or a true eye. This simple definition of the end of a game is sufficient to score a final go position, and to make the method work. Since 2002, various 9x9 MC go programs have been created: Oleg by Helmstetter [7], Vegos by Kaminski [11], Go81 by Raiko [12], DumbGo by Hamlen and Indigo [3]. A heuristic worth considering is the all-move-as-first heuristic that was used by Gobble and Oleg. In addition to updating the mean value of the first move of the random game, the all-move-as-first heuristic updates the mean values of all moves played first on an intersection during the random game [8]. It significantly increases the speed of the program but the playing level is decreased [7].

Until 2002, Indigo remained a classical go program based on local tree search [4], and on extensive knowledge [6]. Since then, it has associated knowledge with a Monte-Carlo approach [5], and has used progressive pruning [2,7] but it does not use the all-move-as-first heuristic. Indigo can be considered as the first 19x19 MC go program. Figure 1 provides an overview of the two-step move decision process used in Indigo in 2003. The pre-selection module gives N_{select} moves, or N_s for short, to the MC module that, in turn, selects the best move among the N_s moves. The work described here heavily depends on this two-step architecture.

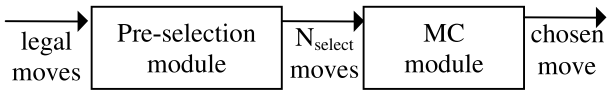

Fig. 1. Indigo architecture is made up with a pre-selection module and a MC module.

This paper aims at assessing two heuristics worth integrating into a MC go program using such an architecture: the History Heuristic (HH) and the Territory Heuristic (TH). HH and TH were successfully integrated in the 2004 release of Indigo. They merely consist in using the MC information output from the MC module as input of the two modules as shown in Figure 2. In addition to the MC evaluation of a position, the TH computes the expected outcome of each intersection of the position. Then TH uses the MC outcomes of intersections to define MC territories, and to refine move urgency within the pre-selection module. The HH uses the past MC evaluations of moves either to remove bad moves from the candidate list of the pre-selection module (see Figure 2), or to lower move urgency within random games. These two heuristics are more precisely defined in section 2 that also describes the current work and its motivations. Section 3 highlights the experiments. Finally, section 4 provides a conclusion.

2. Current Work

First, subsections 2.1 and 2.3 show how to improve the two-step move decision process of figure 1 by performing a preliminary MC evaluation yielding MC territories. TH consists in using the MC territories. Second, because past MC simulations were already performed on the past positions of the current game, subsections 2.4 and 2.5

show how to use the results of the past MC simulations to improve move selection on the current position, which is HH. Each heuristic can be applied in two ways, either external or internal. The term “internal” means it is used within the random game move selection, and “external” means it is used within the current game move preselection. Thus, we get four heuristics: the External Territory Heuristic (ETH), the Internal Territory Heuristic (ITH), the External History Heuristic (EHH), and the Internal History Heuristic (IHH). Figure 2 shows the aim of the external heuristics: using the available MC information within the move pre-selection module.

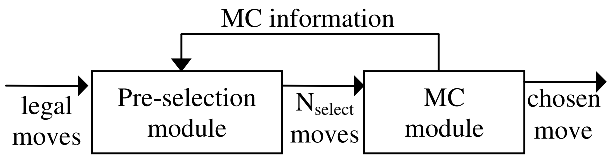

Fig. 2. An external heuristic consists in using the MC feedback within the move pre-selection module.

2.1. *MC evaluation and TH*

In addition to the score of a terminal position, the outcome of each intersection (+1 if controlled by Black, -1 if White) is also available. Thus, in addition to the position MC evaluation (that is the mean of terminal position scores [1]), for each intersection, the mean of the intersection outcome is computable as well. After few hundred of random games, this information is available with a reasonable precision. Hence, it can be used to enhance move selection either within pre-selection (ETH) or within the random games (ITH). A transformation function F from $[-1, +1]$ to $[0, 1]$ yields the probability of playing on an intersection, given its expected outcome. The function $F(x) = exp(-kx^2)$ answers the requirement with $k > 0$ in general. Since ETH and ITH are used in different conditions, we used two variables, k_i and k_e. Concerning the internal heuristic, we set $P_i = exp(-k_i x^2)$ with $k_i > 0$, and for ETH, we set $P_e = exp(-k_e x^2)$. In both cases, the greater the probability of being controlled, the smaller the urgency to play on it.

2.2. *MC evaluation and ETH*

ETH works simply. Before processing the MC search and the move pre-selection, Indigo launches random games to evaluate the current position, and it computes the mean value of each intersection outcome x. After a sufficient number of random games, for each intersection, the probability of playing on it is calculated with $P_e = exp(-k_e x^2)$. Then at the end of move pre-selection, for each intersection, the pre-selection module multiplies the knowledge-based urgency of playing on it by

P_e, yielding the actual urgency of playing on it. Then, the actual urgency of an intersection is used to determine the N_s moves given to MC search. The value of k_e is set such as $exp(-k_e)$ is sufficiently small to make ETH effective, but not too small to avoid strong pruning. $exp(-k_e) = 0.01$ is the value used for the experiments described below.

2.3. *MC evaluation and ITH*

The description of TH can be considered as the first iteration of an iterative process. The first iteration consists in performing random games to estimate the intersection expected outcomes. The next iterations launch random games in which the urgency of a move is multiplied by the probability of playing on its corresponding intersection (defined in subsection 2.1). After a few iterations, one could expect that the MC evaluation and the intersection expected outcomes are better estimated than with the one-iteration method. Next, we call TI the number of territory iterations performed by using ITH.

2.4. *IHH*

IHH consists in updating a history number for each move and in using this number during a random game to select a move. It is analogous to the history heuristic defined by [13] in the game tree framework. Within the MC framework, the urgency of a move played during a random game is multiplied by $exp(Kv)$, in which v is the current evaluation of the move, and K a constant, inverse of the temperature ($K = 0$ means $T = \infty$). This heuristic was already assessed by [7] on 9x9 boards within a pure MC background (very little knowledge). It was also used in [8,11] with a varying temperature. [7] resulted in a +8 improvement on 9x9 boards. In the current work, the architecture is designed for playing on 19x19 boards, and consequently contains a pre-processor that does not allow to update v for all the legal moves, but for the N_s pre-selected moves only. Thus, IHH lies out of the scope of the current architecture. However EHH remains possible.

2.5. *EHH*

A pre-selected move badly ranked by MC on a given position occuring in a game, was very often re-selected and badly re-ranked on the following positions of the game. Therefore, in the flow of a game, the set of pre-selected moves could be filled with moves badly assessed by the MC module. Worse than being pre-selected, these bad moves stayed for a long time in the set of pre-selected moves. As such, the program was not able to get rid of the bad moves from the pre-selection set. The program ran as if N_s was lowered down to 2 or even 1, which has dramatic consequences on the quality of the chosen move.

Hence, it was mandatory to give a penalty to a move badly ranked by the MC module in its previous history. Each move has a history divider whose default value is set up to 1. During pre-selection, the move urgency is set up with the knowledge-based urgency divided by the history divider. For each pre-selected move, the rank determined by the MC module is kept. The history divider of the second worst move ranked by MC is multiplied by B, called the "base". The history divider of the worst move ranked by MC is multiplied by B^2. This way, a bad move is less likely to stay in the pre-selection set. In addition, when a move is actually played in a game, the history divider of the moves situated at a Manhattan distance inferior to a radius R from the actual move, are reset to 1. Any move can thus be reset by the last move, and possibly pre-selected again.

3. Experiments

For each heuristic, we set up experiments to assess their effect on the level of our programs. One experiment consists of a 400-game match between the program to be assessed and the experiment reference program, each program playing 200 games with Black. Given that the standard deviation of 19x19 games played by our programs is roughly 50 points, 400 games enable our experiments to lower σ down to 2.5 points and to obtain a 95% confidence interval of which the radius equals 2σ (i.e. 5 points). We have used 2.4 GHz computers. When the response time of the assessed program varies with the experimental parameters, we mention it. The name of the prototype program is HENRY which stands for HEURISTIC EX(IN)TERNAL TERRI(HIS)TORY. Furthermore, all programs described here do not use any conservative or aggressive style depending on who is ahead in a game, they only try to maximize their own score. The score of a game is more significant than the winning percentage which is consequently not included in the experiments' results.

3.1. *ETH assessment*

In this experiment, we aim at gathering two kinds of results: those obtained in self-play, and those obtained against a reference program. Self-play consists in a confrontation between HENRY using ETH against HENRY not using ETH. The reference program is GNU Go 3.2 [10]. Table 1 shows the improvement brought about by ETH.

Table 1. Improvement brought about by ETH within self-play and against GNU GO.

	self-play	GNU Go
9x9	+1	-1
13x13	+15	+3
19x19	+43	+10

On 19x19 boards, the improvement observed in self-play cannot be overlooked. It is important from a go perspective. However, against GNU Go 3.2 which is differently designed, its importance is less obvious than within self-play. ETH was integrated in the 2004 release of Indigo that played the 2004 Olympiads and won the bronze medal [9].

3.2. *ITH assessment*

Henry has two parameters: TI, the number of territory iterations, and P_i, the probability of playing on intersections controlled by one player (defined in subsection 2.1). Henry$(TI = 1, P_i = 1)$ is thus the reference release. We aim at finding out the best values of TI and P_i. TI may equal 2, 3, 4 or 5. We performed three tests. The results of Henry$(TI, P_i = 0.1)$ vs Henry$(TI = 1, P_i = 0.1)$ are shown in Table 2.

Table 2. Henry$(TI = 2, 3, 4, P_i = 0.1)$ vs Henry$(TI = 1, P_i = 0.1)$.

	2	3	4
9x9	-3	+1	-2
13x13	-1	+1	-4
19x19	-7	-1	+5

Since most of the results are inferior to 5 points, no conclusion can be drawn from Table 2 with sufficient statistical confidence. The results of Henry$(TI, P_i = 0.4)$ vs Henry$(TI = 1, P_i = 0.4)$ are given in Table 3.

Table 3. Henry$(TI = 2, 3, 4, P_i = 0.4)$ vs Henry$(TI = 1, P_i = 0.4)$.

	2	3	4
9x9	+3	+2	0
13x13	+2	+3	+1
19x19	+3	+6	+7

The results given in Table 3 are slightly positive but, again, no certain statistical conclusion can be drawn for most cells. However, by improving the reference program by 6 points with a 10% time overhead, Henry$(TI = 3, P_i = 0.4)$, can offer a good compromise between move quality and time. The results of Henry$(TI, P_i = 0.4)$ vs Henry$(TI, P_i = 0.1)$ are provided by Table 4.

Table 4. HENRY($TI = 2, 3, P_i = 0.4$) vs HENRY($TI = 2, 3, P_i = 0.1$).

	2	3
9x9	-0.5	-0.8
13x13	-2.6	-3.3
19x19	-3.6	+1.6

Since all the results are inferior to 5 points, Table 4 shows that HENRY($TI = 2, 3, P_i = 0.4$) and HENRY($TI = 2, 3, P_i = 0.1$) have approximately the same strength. We think that, provided P_i is low, its value has no great importance on move quality. Finally, summing up the results from Tables 2, 3 and 4 to find out a statistical conclusion remains difficult. Nevertheless, taking the small time overhead (10%) due to processing territory iterations, and using ITH with ($TI = 3, P_i = 0.4$) can be worth considering on 19x19 boards (about 6-point improvement) but not really on 9x9 or 13x13 boards.

3.3. *EHH assessement*

In this subsection, HENRY has two parameters: B and R. HENRY($B = 1, R = 19$) is the reference release. In a first stage, we aim at roughly finding out the "best" values of B and R with B equalling 2, 5 or 10 and $5 <= R <= 13$. Because this was a first stage experiment, we did 100 games per confrontation only to obtain approximate results. The experiment is performed on 19x19 boards. Table 5 yields the results showing that $B = 2$ remains the best choice.

Table 5. HENRY($B = 2, 5, 10, 5 <= R <= 13$) vs HENRY($B = 1, R = 19$).

	5	7	9	11	13
2	+9	+8	+7	+4	0
5	0	+4	-1	-3	+3
10	-6	-5	-4	-10	-8

Then, in a second stage with $B = 2$, we aimed at determining more precisely the best value of R, performing 400 games per confrontation again. Table 6 yields the results. Due to a different number of games per confrontation, the results given in Table 6 are more accurate than the results given in Table 5.

The ($B = 2, R = 4$) result is amazing. It shows a discontinuity that we are not able to explain. HENRY($B = 2, R = 7$) gives the best improvement (+12 points),

Table 6. HENRY($B = 2, 2 <= R <= 9$) vs HENRY($B = 1, R = 19$).

	2	3	4	5	6	7	8	9
mean	+5	+9	0	+7	+9	+12	+9	+5

which is significant but not extraordinary on go standards. Against GNU Go 3.2, the improvement is about +5 points, smaller than in self-play. Since the best value of B is the smallest tried, trying a smaller value like 1.5 is a remaining perspective.

4. Conclusion

In this paper, we have defined the four following heuristics: IHH, EHH, ITH and ETH within the Monte-Carlo go framework, and we have experimentally evaluated the playing level improvement brought about by their integration within INDIGO, either in self-play or against GNU Go. ETH is the most effective heuristic, enabling INDIGO to improve by 40 points on 19x19 boards in self-play without time overhead. The improvement brought about by ETH is smaller against GNU Go (+10 points) but still significant. ITH is interesting but not effective enough (+5 points) to compensate the time overhead (+10%). IHH cannot be taken into account in our pre-selection framework. Finally, EHH brought about a 12-point improvement within self-play, and a 5-point improvement against GNU Go. The external heuristics use the information processed by the MC module as a feedback within the pre-selection module as shown in Figure 2. These heuristics enabled Indigo to perform well (bronze medal) at the 2004 Computer Olympiad on 19x19 boards [9]. Regarding the scope of Monte Carlo go, we have several interesting perspectives: re-produce the hints of [15], use reinforcement learning [16] to improve the quality of move urgencies in random games, and study local MC search.

References

1. Abramson, B.: Expected-outcome : a general model of static evaluation. IEEE Transactions on PAMI, **12** (1990) 182–193
2. Billings, D., Davidson, A., Schaeffer, J., Szafron, D.: The challenge of poker. *Artificial Intelligence*, **134** (2002) 201–240
3. Bouzy, B.: Indigo home page. www.math-info.univ-paris5.fr/∼bouzy/INDIGO.html (2005)
4. Bouzy, B.: The move decision process of Indigo, *International Computer Game Association Journal*, **26**, 1, (2003) 14–27
5. Bouzy B.: Associating knowledge and Monte-Carlo approaches within a Go program. *Information Sciences*, **175**,(2005) 247–257
6. Bouzy, B., Cazenave, T.: Computer Go: an AI oriented survey. *Artificial Intelligence*, (2001) **132** 39–103
7. Bouzy, B., Helmstetter, B.: Monte Carlo Go Developments. *Advances in Computer Games, Many Games, Many Challenges*, J. van den Herik, H. Iida, E. Heinz (eds) (2003) 159–174

8. Brügmann, B.: Monte Carlo Go. www.joy.ne.jp/welcome/igs/Go/computer/-mcgo.tex.Z (1993)
9. Fotland, D.: Go Intellect wins 19x19 Go tournament, *International Computer Game Association Journal*, (2004) **27** 3 169–170
10. Free Software Foundation: GNU Go home page, www.gnu.org/software/gnugo/, (2005)
11. P. Kaminski: Vegos home page, www.ideanest.com/vegos/, (2003)
12. Raiko, T.: The Go playing program called GO81. *FAIC conference proc.*, Helsinki (2004) 197–206
13. J. Schaeffer: The history heuristic and Alpha-Beta Search Enhancements in Practice, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1989), **11**, 11, 1203–1212
14. Sheppard, B.: World-championship-caliber Scrabble. *Artificial Intelligence*, (2002) **134** 241–275
15. Sheppard, B.: Effective Control of Selective Simulation. *International Computer Game Association Journal*, (2004) **27** 2 67–80
16. R. Sutton and A. Barto: *Reinforcement Learning: an introduction*, MIT Press (1998)
17. Tesauro, G., Galperin, G.: On-line policy improvement using Monte-Carlo search, *Advances in Neural Information Processing Systems*, MIT Press (1996) 1068–1074