Associating domain-dependent knowledge and Monte Carlo approaches within a go program

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,
tél: (33) (0)1 44 55 35 58, fax: (33) (0)1 44 55 35 35,
email: bouzy@math-info.univ-paris5.fr,
http: www.math-info.univ-paris5.fr/∼bouzy/

Abstract

This paper underlines the association of two computer go approaches, a domain-dependent knowledge approach and Monte Carlo. First, the strengthes and weaknesses of the two existing approaches are related. Then, the association is described in two steps. A first step consists in using domain-dependent knowledge within the random games enabling the program to compute evaluations that are more significant than before. A second step simply lies in pre-processing the Monte Carlo process with a knowledge-based move generator in order to speed up the program and to eliminate tactically bad moves. We set up experiments demonstrating the relevance of this association, used by Indigo at the 8th computer olympiad as well.

1 Introduction

Over the past years, we have improved our go program Indigo [6] by starting from the previous year's version, considering its main defects and trying to supply remedies. As Indigo is largely based on domain-dependent knowledge, it has become more and more difficult to improve. Thus, in 2002, we tried different approaches to build Olga, a *very little* knowledge go program based on Monte Carlo simulations. Interestingly, Olga contains very little go knowledge, and, yet, can be situated on a par with Indigo containing a lot of go knowledge on 9x9 boards [10]. Consequently, it is worthwhile to assess the level of a program that uses both domain-dependent knowledge *and* Monte Carlo approaches, which is the aim of this paper.

In this aim, section 2 describes the related work: Indigo, a domain-dependent knowledge approach, and existing Monte Carlo approaches. Then, section 3 focuses on the description of the programs to be assessed. Section 4 highlights the results which show that it is possible to successfully associate domain-dependent knowledge and Monte Carlo approaches within a more efficient go program than the early ones.

2 Related Work

In this section, we put the emphasis on the strong and weak points of Indigo, our domain-dependent knowledge based program, and on already existing Monte Carlo go programs.

2.1 A knowledge based approach

Indigo [6, 5] is a classical go program based on tree search [8] and on extensive knowledge [9]. For instance, territories and influence are modelled by means of the mathematical morphology [7]. As most domain-dependent knowledge go programs, Indigo's weaknesses lie in a weak global sense that results from the breaking up of the whole problem into sub-problems. Furthermore, some holes in the knowledge remain difficult to cover because of interactions between the various elements of knowledge. Fortunately, relative to its level, Indigo has its strengthes, such as fighting by using adequate rules to this end, and tactical ability by using tree search.

2.2 Monte Carlo Go approaches

The existing work about Monte Carlo simulations applied to computer go is [11, 14] and recently [10]. [11, 14], based on simulated annealing [15], should be more appropriately named simulated annealing go. [10] is a recent study of Monte Carlo approaches in its general meaning - using the computer random function and averaging the results of episodes. The basic idea is: to evaluate a position by playing a given number of completely random games to the end - without filling the eyes - and then scoring 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. [10] experimentally proves the superiority of progressive pruning over simulated annealing. Progressive pruning is based on [2, 3], and was used in [19]. Each move has a mean value m, a standard deviation σ, a left expected outcome m_l and a right expected outcome m_r. For a move, $m_l = m - \sigma r_d$ and $m_r = m + \sigma r_d$. r_d is called the ratio for difference. A move M_1 is said to be statistically inferior to another move M_2 if $M_1.m_r < M_2.m_l$. (The dot means the access to the slot of a data structure). Two moves M_1 and M_2 are statistically equal when $M_1.\sigma<\sigma_e$ and $M_2.\sigma<\sigma_e$ and no move is statistically inferior to the other. σ_e is called standard deviation for equality. After a

minimal number (N_m times the number of legal moves) of random games, a move is pruned as soon as it is statistically inferior to another move. Therefore, the number of candidate moves decreases while the process is running. The process stops either when there is only one move left (this move is selected), when the moves left are statistically equal, or when a maximal threshold of iterations is reached (N_m^2 times the number of legal moves). In these two cases, the move with the highest expected outcome is chosen. Independently of the use of progressive pruning, Monte Carlo based go programs such as Olga have a good global sense but a weak tactical ability.

3 Our Work

Given that knowledge based go programs such as Indigo have a good tactical ability, and that Monte Carlo go programs such as Olga have a good global sense, it appeared logical to develop a go program that uses both knowledge and Monte Carlo simulations to obtain the best of both worlds: a good tactical ability and a good global sense. From the Monte Carlo viewpoint, starting from Olga(pseudo = false, preprocess = false), we have built two programs. First, we replaced the uniform probability based random move generator by a pseudo-random move generator using little go knowledge, which yielded Olga(pseudo = true, preprocess = false). Second, we speeded up and enhanced this program by preprocessing it with a knowledge based move generator available in Indigo, which brought about Olga(pseudo = true, preprocess = true).

3.1 Pseudo-random move generation

Olga(pseudo = true) uses pseudo-random game simulations. The principle underlying move generation in Olga(pseudo = true) is almost the same as presented in section 2.2. The difference lies in the way the moves are generated within the random games. Instead of generating the move according a uniform probability, Olga(pseudo = true) generates moves according a probability dependent on go knowledge. To choose a move within a random game, Olga(pseudo = true) uses move urgencies, and the probability to choose a move is then linear in the move urgency. The problem is to correctly define the move urgencies. Olga(pseudo = true) uses rules about string captures and 3x3 patterns to obtain the move urgencies.

On the one hand, a string with one liberty only, results in a very great urgency to the move that captures the string. In this case the urgency is linear in the string size. On the other hand, all the very small patterns of Indigo, that are included in a 3x3 window centered around a move, are used to build a small database of 3x3 patterns. Each pattern advises the random move generator to play the move situated in its center with an urgency accessed in a table. When neither the edges of the board nor the symmetries and rotations are taken into account, there are only 3^8 patterns of this kind. Taking the edges into account, multiplies this number by 25 at most. Time constraints make it impossible to

consider symmetries and rotations. Nevertheless, it is easy to set up a table of move urgencies in the memory of the computer whose access is direct in the 3x3 bit set around the move.

The size of patterns is very small due to time constraints. With 3x3 patterns, the simulation time is acceptable: twice as slow as the uniform probability based simulation. For instance, on a 1.7 GHz computer, Olga with string and pattern urgencies plays 3,000 random 9x9 games per second.

With such go knowledge, the pseudo-random games are more plausible games than completely random games, which gives a better approximation of the position evaluation. The remaining problem lies in the presence of bias while building the move urgencies. In this context, we reuse the Indigo pattern database, which has been tuned for several years and whose urgencies are, if not optimized, acceptable.

The standard deviation σ of the pseudo-random games is roughly the same as the standard deviation of the simple random games: about 35 points on 9x9 boards, 50 points on 13x13 boards and about 70 points on 19x19 boards. Consequently, the number of pseudo-random games necessary to obtain a given precision with Olga(pseudo = true) remains the same as in Olga(pseudo = false). On 9x9 boards, 1000 games enable our experiments to lower σ down to 1 point and to obtain a 95% confidence interval of which the radius equals 2 points.

3.2 Preprocessing with knowledge

Since Olga(pseudo = true, preprocess = false) does not use any tree search, it stays weak tactically. Furthermore, because Monte Carlo simulations are very expensive to compute with a sufficient precision, this program spends one full day to play a 19x19 game on a 1.7 Ghz computer. Therefore, to overcome these two downsides in one move, we added Indigo's move generator to Olga as a preprocessor of simulations. This preprocessor selects the N_s best moves and gives them to the Monte Carlo module that chooses the best move. Obviously, the tactically bad moves are eliminated by the preprocessor and a small value of N_s enables Olga(pseudo = true, preprocess = true) to complete a 19x19 game in a reasonable time.

4 Experiments

This section provides the results of the experiments carried out until now in a chronological way. Because our initial aim was to improve Indigo2002, we first present the result of Olga(pseudo = true, preprocess = false) against Indigo (subsection 4.1) and Olga(pseudo = true, preprocess = true) against Indigo (subsection 4.2). Then, to highlight the positive effect of knowledge within random games, we show the result of Olga(pseudo = true) against Olga(pseudo = false) in subsection 4.3. In subsection 4.4, we assess our work with a confrontation against a very differently designed program, GNU Go, the well-known go playing program of the FSF [12]. Finally, in subsection 4.5, to give an idea of

how a Monte Carlo program plays, we show a 9x9 game between Olga(pseudo = true, preprocess = true) and its author.

One confrontation consists in a match of 100 games between 2 programs, each program playing 50 games with Black. The result of such a confrontation is the mean score and a winning percentage when the number of games performed is sufficient. Given that the standard deviation of games played on 9x9 boards (respectively 13x13 and 19x19 boards) is roughly 15 points (respectively 25 and 40), 100 games enable our experiments to lower σ down to 1.5 point (respectively 2.5 points and 4 points) and to obtain a 95% confidence interval. We have used 1.7 GHz computers, and we mention the response time of each program. The variety of games is guaranteed by the different random seeds of each run of Olga, Indigo and GNU Go.

4.1 Olga(pseudo = true, preprocess = false) vs Indigo

During the first stage of our tests, we set up games between Olga(pseudo = true, preprocess = false) and Indigo2002 on 9x9, 13x13 and 19x19 boards. Table 1 shows the results on Olga's side (+ means a win for Olga).

board size	9x9	13x13	19x19
mean	+12	+24	+45
time	20'	2h30'	20h
games	20	20	1

Table 1: Results of Olga(pseudo = true, preprocess = false, $r_d = 1.0$, $\sigma_e = 0.4$) against Indigo2002 for the usual board sizes

On 9x9, while Olga(pseudo = false) matches Indigo [10], Olga(pseudo = true) is about 12 points better than Indigo. On 13x13, while Olga(pseudo = false) is 20 points worse than Indigo [10], Olga(pseudo = true) is 24 points better. This board size is the appropriate one to underline the strength of Olga(pseudo = true). Due to the length of the game on 19x19 boards, we set up only one game in which Olga(pseudo = true) playing black wins with 45 points. This game highlights the very different styles of programs rather than the quantitative result. Olga plays very well globally by circling large areas, and killing groups whenever it is possible. Thanks to its tactical strength, Indigo collects points, and takes advantage of Olga's blind point in tactics. Of course, due to the very low number of games performed, the results of table 1 are not statistically significant.

4.2 Olga(pseudo = true, preprocess = true) vs Indigo

In this set of experiments, we assess Olga(pseudo = true, preprocess = true) against Indigo 2002 in time and level with the three classical board sizes. Table 2 provides the results of Olga(pseudo = true, preprocess = true) against Indigo2002.

board size	9x9	13x13	19x19
mean	+18	+35	+44
% wins	76%	88%	75%
time	1'30"	10'	1h30'

Table 2: Results of Olga(pseudo = true, preprocess = true) against Indigo2002 for the usual board sizes, $N_s = 10$, $N_m = 50$, $r_d = 1.0$, $\sigma_e = 0.4$

Whatever the size of the board, the results of Olga are excellent against Indigo2002. On 9x9 boards, the mean score is high (+18) while the standard deviation is also high, resulting in a "weak" winning percentage (76% only). On 13x13 boards, Olga obtains her best winning percentage. However, table 2 does not shed the light on important parameters for controlling both time and level of the program. These parameters are N_m, r_d, and N_s. In our view, N_s is the most important parameter, and table 3 shows the results in N_s on 19x19 boards.

N_s	2	4	7	10	15	20
mean	-7	+18	+25	+44	+56	+68
% wins	41%	59%	66%	75%	90%	91%
time	15'	40'	1h10'	1h30'	2h	2h30'

Table 3: Results of Olga(pseudo = true, preprocess = true, $N_m = 50$, $r_d = 1.0$, $\sigma_e = 0.4$) against Indigo2002 for N_s varying from 2 up to 20, with 19x19 boards.

Olga(pseudo = true, $N_s = 1$) corresponds to the urgent method [8] of Indigo2002 selecting one move without verification. Its level is necessarily inferior to the one of Indigo2002 that uses a calm method in addition to the urgent method with verification [8]. Thus, its entry is not mentioned in the table. Olga(pseudo = true, $N_s = 2$) selecting two moves with Indigo2002's urgent method while choosing the best one by running pseudo-random game simulations, has a great similarity to Indigo2002's urgent method. This explains the almost zero mean when $N_s = 2$. Olga($N_s = 4$) and Olga($N_s = 7$) are interesting as they play significantly better on average than Indigo2002 and their execution time is suitable on 1.7 Ghz computers. With more computing power, N_s can be higher and Olga($N_s = 10$, 15, 20), then, gives good results. Moreover, other experiments carried out with other values for N_m, r_d, and σ_e show that $N_m < 25$ is not acceptable, and that $r_d > 1.0$ is mandatory. σ_e has not much importance; its value can be lowered to 0.2 to obtain slightly better results.

4.3 Olga(pseudo = true, preprocess = true) vs Olga(pseudo = false, preprocess = true)

By preprocessing, the time used by Monte Carlo programs becomes reasonable. Thus, the experiment assessing the use of domain-dependent knowledge within the random games can be carried out on 13x13 and 19x19 boards. Table 4 provides the results of Olga(pseudo = true, preprocess = true) against Olga(pseudo = false, preprocess = true).

board size	9x9	13x13	19x19
mean	+8	+40	+100
% wins	68%	93%	97%

Table 4: Results of Olga(pseudo = true) against Olga(pseudo = false) for the usual board sizes, preprocess = true, $N_s = 10$, $N_m = 50$, $r_d = 1.0$, $\sigma_e = 0.4$

These results are self-explanatory. They clearly prove that the program using pseudo-random games is significantly better than the program using uniform probability based random games. The greater the size of the board, the greater the difference between the two programs. On 19x19 boards, the difference reaches one hundred points in average, which is huge for go standards.

4.4 Olga(pseudo = true, preprocess = true) vs GNU Go-3.2

This section shows the result of Olga(pseudo = true, preprocess = true) against GNU Go [12]. We choose GNU Go-3.2 with its default level. Needless to say, GNU Go remains superior to Olga(pseudo = true, preprocess = true): the minus signs on the "mean" line of table 5 indicate the superiority of GNU Go over both Indigo and Olga. But in order to highlight the improvement brought about by the association of knowledge and statistics over knowledge only, table 5 provides Indigo2002's result on the left part, and Olga's one on the right part. The result is shown for each classical size.

	Indigo2002			Olga(true, true)		
board size	9x9	13x13	19x19	9x9	13x13	19x19
% wins	35%	13%	6%	37%	33%	19%
mean	-9	-34	-83	-5	-15	-40
time	20"	1'	2'	12'	1h	5h

Table 5: Results of Indigo2002 and Olga(pseudo = true, preprocess = true, $N_s = 10$, $N_m = 100$, $r_d = 2.0$, $\sigma_e = 0.4$) against GNU Go-3.2.

On 9x9 boards, the improvement is not striking: Olga performs four points better than Indigo only. But, it is important to notice the improvement induced

by the addition of knowledge within random games: Olga(pseudo = true, preprocess = true) is only five points worse than GNU Go while Olga(pseudo = false, preprocess = false) is 34 points worse than GNU Go-3.2 [10]. Furthermore, the improvement on 13x13 and 19x19 boards is worth being underlined: the gap between Indigo and GNU Go is reduced by a half, which is a very promising result. The cost of this improvement lies in the response time: with $N_s = 10$, $N_m = 100$, $r_d = 2.0$, Olga spends 5 hours to play one 19x19 game, while Indigo plays out a full 19x19 game in a couple of minutes.

4.5 Olga(pseudo = true, preprocess = true) vs its author

Figure 1 shows a game between Olga (pseudo = true, preprocess = true) playing Black and its author playing White. White played calmly not to crush the program. In this context, Olga often played good and safe moves. Black 19, 21, 23 were the first strange moves. They uncovered a feature of Monte Carlo programs: threatening the opponent even if the sequence does not work. At least, the opponent answered and the program kept the initiative. Black 31 was the second mistake, always threatening something but finally loosing Black 27. In the endgame, Black lost its upper left corner but played safely to keep its group alive.

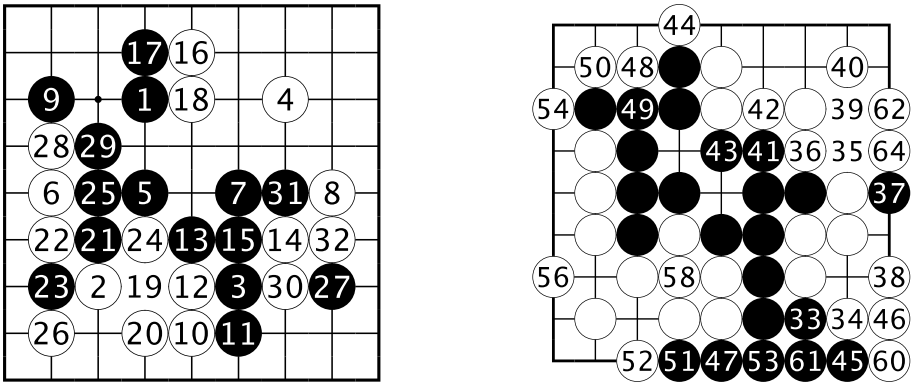

Figure 1: Olga(Black)-Bouzy(White). White wins by 33 points on the board.

5 Conclusion and perspectives

Starting from Indigo2002, a domain-dependent knowledge and tree search based program, we set up a new go program, Olga(pseudo = true, preprocess = true) that associates this domain-dependent knowledge with a Monte Carlo approach. First, local knowledge is used efficiently to yield the non-uniform probability to moves within pseudo-random games. Second, a lot of knowledge is used to filter the moves provided to Monte Carlo simulations, and thereby, avoiding tactical blunders.

Table 6 summarizes the results of the confrontations performed between Olga(pseudo = false, preprocess = true), Olga(pseudo = true, preprocess = true), Indigo2002 and GNU Go-3.2 for each classical size, assuming that the program of the column is the max player.

	Indigo2002			Olga(true, true)		
board size	9x9	13x13	19x19	9x9	13x13	19x19
Olga(false, true)	-2	+3	+12	+8	+40	+100
Indigo2002				+18	+34	+44
GNUGo-3.2	-9	-34	-83	-5	-15	-40

Table 6: Summary of confrontations between Olga, Indigo and GNU Go for each classical board size.

First, table 6 mentions that Olga(pseudo = false, preprocess = true) can be situated on a par with Indigo2002. Then, it turns out that Olga(pseudo = true, preprocess = true) is significantly stronger than Indigo2002 but still weaker than GNUGo-3.2. On 19x19 boards and under reasonable time constraints (one hour and a half), Olga(pseudo = true, preprocess = true) ranks about forty points better than Indigo2002, and one hundred points better than Olga(pseudo = false, preprocess = true). For 2003, this constitutes a significant improvement. In such a context, we may say that pseudo-random Monte Carlo simulations provide the 2003 remedy to Indigo2002's weaknesses. To attend the 2003 computer olympiad in Graz [1], Indigo2003 was built by merging Indigo2002 and Olga(pseudo = true, preprocess = true). Indigo2003 ranked 5th upon 11 programs in the 19x19 competition [13] and 4th upon 10 programs in the 9x9 competition [20], thus confirming our idea that associating knowledge and Monte Carlo is appropriate to computer go.

Considering the ever-increasing power and memory of computers, merely increasing the size of patterns for pseudo-random games to greater shapes will surely be relevant in the near future. From the statistical angle, the main perspective is to generate both the pattern database crucial to preprocessing and the pattern database for pseudo-random games, both in an automatic manner by using games available on the Internet as advised by [17]. For instance, the assessment of how well the 3x3 patterns are at picking good moves could be done with a bayesian approach by using professional games [4]. Another possibility to improve the adequacy of move urgencies within random games is reinforcement learning [18], or more specifically Q-learning [21]. If a remedy can be found to speed up both Indigo evaluation function and move generator, then another experiment worth considering would be to replace the full random games by shallow sequences of moves generated by Indigo and then call the Indigo conceptual evaluation function. Such an experiment has been performed with success at Backgammon [19] with "truncated rollouts" using a parallel approach. From the tactical angle, upgrading our depth-one approach with a best-first tree search [16] is worthwhile to be integrated within the current work.

Actually, we already integrated a depth-three global tree search within our 9x9 release that attended the 2003 computer olympiad. Finally, from the practical angle, determining the best values of the relevant parameters (N_s, N_m, r_d, size of patterns) under time constraints cannot be overlooked, and will require more experiments.

References

[1] 8th computer olympiad home page. www.cs.unimaas.nl/Olympiad2003/, 2003.

[2] B. Abramson. Expected-outcome : a general model of static evaluation. *IEEE Transactions on PAMI*, 12:182–193, 1990.

[3] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134:201–240, 2002.

[4] C. Bishop. *Neural networks and pattern recognition*. Oxford University Press, 1995.

[5] B. Bouzy. The indigo program. In *2nd Game Programming Workshop in Japan*, pages 197–206, Hakone, 1995.

[6] B. Bouzy. Indigo home page. www.math-info.univ-paris5.fr/∼bouzy/INDIGO.html, 2002.

[7] B. Bouzy. Mathematical morphology applied to computer go. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(2):257–268, March 2003.

[8] B. Bouzy. The move decision process of indigo. *International Computer Game Association Journal*, 26(1):14–27, March 2003.

[9] B. Bouzy and T. Cazenave. Computer go: an ai oriented survey. *Artificial Intelligence*, 132:39–103, 2001.

[10] B. Bouzy and B. Helmstetter. Monte carlo go developments. In Ernst A. Heinz H. Jaap van den Herik, Hiroyuki Iida, editor, *10th Advances in Computer Games*, pages 159–174, Graz, 2003. Kluwer Academic Publishers.

[11] B. Bruegmann. Monte carlo go. www.joy.ne.jp/welcome/igs/Go/computer/mcgo.tex.Z, 1993.

[12] D. Bump. Gnugo home page. www.gnu.org/software/gnugo/devel.html, 2003.

[13] K. Chen. Gnu go wins 19x19 go tournament. *International Computer Game Association Journal*, 26(4):261–262, December 2003.

[14] P. Kaminski. Vegos home page. www.ideanest.com/vegos/, 2003.

[15] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, May 1983.

[16] R. Korf and D. Chickering. Best-first search. *Artificial Intelligence*, 84:299–337, 1994.

[17] N. Schraudolf, N. Dayan, and T. Sejnowski. Temporal difference learning of a position evaluation in the game of go. In Cowan, Tesauro, and Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 817–824. Morgan Kaufmann, San Francisco, 1994.

[18] R. Sutton and A. Barto. *Reinforcement Learning: an introduction*. MIT Press, 1998.

[19] G. Tesauro and G. Galperin. On-line policy improvement using monte carlo search. In *Advances in Neural Information Processing Systems*, pages 1068–1074, Cambridge MA, 1996. MIT Press.

[20] E. van der Werf. Aya wins 9x9 go tournament. *International Computer Game Association Journal*, 26(4):263, December 2003.

[21] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.