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/

1 Introduction

Over the past years, we have improved Indigo [4] our go program 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 [8]. Consequently, it is worthwhile to assess the level of a program that uses both domain-dependent knowledge *and* Monte Carlo approaches. In this paper, 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. Before conclusion, section 4 highlights the results which show that it is possible to 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 [4, 3] is a classical go program based on tree search [6] and on extensive knowledge [7]. For instance, territories and influence are modelled by means of the mathematical morphology [5]. 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 subproblems. 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 [9, 10] and recently [8]. [9, 10], based on simulated annealing [11], should be more appropriately named simulated annealing go. [8] is a recent study of Monte Carlo approaches in its general meaning - using the computer random function. It experimentally proves the superiority of progressive pruning over simulated annealing. Progressive pruning is based on [1, 2]. 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 $M_1.m_r < M_2.m_l$. 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 pseudo random function is obtained by adding rules about string captures and patterns to the random function. On the one hand, a string with one liberty only, results in a very great probability to the move that captures the string. 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 21 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. This way, the simulation time is acceptable: twice as slow as the uniform probability based simulation. 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 of the pseudo-random game mean is roughly the same as the standard deviation of the simple random games. 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).

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 matches assessing Olga(pseudo = true, preprocess = false) and Olga(pseudo = true, preprocess = true) against Indigo.

4.1 Pseudo-random based vs Indigo

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: Time and relative score 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 [8], Olga(pseudo = true) playing black wins all the games, and Olga(pseudo = true) playing white wins half of the games. On 13x13, while Olga(pseudo = false) is twenty points worse than Indigo [8], Olga(pseudo = true) playing black wins 90% of the games, and 70% when playing white. 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 cercling 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.

4.2 Knowledge and pseudo-random based vs Indigo

In this set of experiments, we assess Olga(pseudo = true, preprocess = true) against Indigo 2002 in time and level on 19x19 games. The relevant parameters for controlling both time and level are N_m, r_d, and N_s. Table 2 shows the results in N_s.

N_s	2	4	7	10	15	20
mean	-2	+19	+25	+55	+62	+85
time	15'	40'	1h10'	1h30'	2h15'	2h45'
games	80	80	80	25	18	17

Table 2: Time and relative score 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.

Olga(pseudo = true, $N_s = 1$) corresponds to the urgent method [6] 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 [6]. 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.

5 Conclusion

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.

On 19x19 boards and under reasonable time constraints, this program ranks twenty points better on average than Indigo2002. 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. Indigo2003 will be built by merging Indigo2002 and Olga.

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, in an automatic manner by using games available on the Internet as advised by [12]. Lastly, increasing the size of patterns for pseudo-random games to greater shapes will be relevant, considering the ever-increasing power of computers.

References

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

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

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

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

[5] B. Bouzy. Mathematical morphology applied to computer go. *International Journal of Pattern Recognition and Artificial Intelligence vol 17 n2*, March 2003.

[6] B. Bouzy. The move decision process of indigo. *ICGA Journal vol 25 n1*, March 2003.

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

[8] B. Bouzy and B. Helmestetter. Developments on monte carlo go. www.math-info.univ-paris5.fr/~bouzy/publications.html, 2003.

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

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

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

[12] 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.