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

Bruno Bouzy *

UFR de Mathématiques et d’Informatique, C.R.I.P. 5, Université Paris 5, 45, rue des Saints-Pères, 75270 Paris Cedex 06, France

Accepted 26 April 2004

Abstract

This paper underlines the association of two computer go approaches, a domain-dependent knowledge approach and Monte Carlo. First, the strengths 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.

© 2004 Elsevier Inc. All rights reserved.
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 9×9 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 strengths, 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 $\sigma$, 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 $M_1 \cdot m_r < M_2 \cdot m_l$. (The dot means the access to the slot of a data structure). Two moves $M_1$ and $M_2$ are statistically equal when $M_1 \cdot \sigma < \sigma_e$ and $M_2 \cdot \sigma < \sigma_e$ and no move is statistically inferior to the other. $\sigma_e$ 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 3 \times 3 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 3 \times 3 window centered around a move, are used to build a small database of 3 \times 3 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 3 \times 3 bit set around the move.

The size of patterns is very small due to time constraints. With 3 \times 3 patterns, the simulation time is acceptable: twice as slow as the uniform probability based simulation. For instance, on a 1.7GHz computer, Olga with string and pattern urgencies plays 3000 random 9 \times 9 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 \( \sigma \) of the pseudo-random games is roughly the same as the standard deviation of the simple random games: about 35 points on 9 \times 9 boards, 50 points on 13 \times 13 boards and about 70 points on 19 \times 19 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 9 \times 9 boards, 1000 games enable our experiments to lower \( \sigma \) 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 $19 \times 19$ game on a 1.7GHz 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 $19 \times 19$ 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 (Section 4.1) and Olga (pseudo = true, preprocess = true) against Indigo (Section 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 Section 4.3. In Section 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 Section 4.5, to give an idea of how a Monte Carlo program plays, we show a $9 \times 9$ 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 $9 \times 9$ boards (respectively $13 \times 13$ and $19 \times 19$ boards) is roughly 15 points (respectively 25 and 40), 100 games enable our experiments to lower $\sigma$ down to 1.5 point (respectively 2.5 points and 4 points) and to obtain a 95% confidence interval. We have used 1.7GHz 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 $9 \times 9$, $13 \times 13$ and $19 \times 19$ boards. Table 1 shows the results on Olga’s side (+ means a win for Olga).
On 9 × 9, while Olga (pseudo = false) matches Indigo [10], Olga (pseudo = true) is about 12 points better than Indigo. On 13 × 13, 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 19 × 19 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 Indigo 2002.

Whatever the size of the board, the results of Olga are excellent against Indigo 2002. On 9 × 9 boards, the mean score is high (+18) while the standard deviation is also high, resulting in a “weak” winning percentage (76% only). On 13 × 13 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,

| Table 1 |
|---|---|---|
| Results of Olga (pseudo = true, preprocess = false, $r_d = 1.0$, $\sigma_c = 0.4$) against Indigo 2002 for the usual board sizes |
| Board size | $9 \times 9$ | $13 \times 13$ | $19 \times 19$ |
| Mean | +12 | +24 | +45 |
| Time | 20′ | 2 h 30′ | 20 h |
| Games | 20 | 20 | 1 |

| Table 2 |
|---|---|---|
| Results of Olga (pseudo = true, preprocess = true) against Indigo 2002 for the usual board sizes, $N_s = 10$, $N_m = 50$, $r_d = 1.0$, $\sigma_c = 0.4$ |
| Board size | $9 \times 9$ | $13 \times 13$ | $19 \times 19$ |
| Mean | +18 | +35 | +44 |
| % wins | 76% | 88% | 75% |
| Time | 1′30′ | 10′ | 1 h 30′ |
\( N_s \) is the most important parameter, and Table 3 shows the results in \( N_s \) on 19 \( \times \) 19 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 \( r_e \) show that \( N_m < 25 \) is not acceptable, and that \( r_d > 1.0 \) is mandatory. \( r_e \) 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 13 \( \times \) 13 and 19 \( \times \) 19 boards. Table 4 provides the results of Olga(pseudo = true, preprocess = true) against Olga(pseudo = false, preprocess = true).

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,

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
\( N_s \) & 2 & 4 & 7 & 10 & 15 & 20 \\
\hline
Mean & \( -7 \) & +18 & +25 & +44 & +56 & +68 \\
% wins & 41\% & 59\% & 66\% & 75\% & 90\% & 91\% \\
Time & 15’ & 40’ & 1 h 10’ & 1 h 30’ & 2 h & 2 h 30’ \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Board size & 9 \( \times \) 9 & 13 \( \times \) 13 & 19 \( \times \) 19 \\
\hline
Mean & +8 & +40 & +100 \\
% wins & 68\% & 93\% & 97\% \\
\hline
\end{tabular}
\end{table}
the greater the difference between the two programs. On 19×19 boards, the
difference reaches one hundred points in average, which is huge for go
standards.

4.4. Olga\((pseudo = true, \text{ preprocess} = true) vs GNU Go-3.2\)

This section shows the result of Olga\((pseudo = true, \text{ preprocess} = true)\)
against GNU Go [12]. We choose GNU Go-3.2 with its default level. Needless
to say, GNU Go remains superior to Olga\((pseudo = true, \text{ preprocess} = true)\):
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 improve-
ment brought about by the association of knowledge and statistics over knowl-
edge 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.

On 9×9 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\((pseudo = true, \text{ pre-
process} = true)\) is only five points worse than GNU Go while Olga\((pseudo =
false, \text{ preprocess} = false)\) is 34 points worse than GNU Go-3.2 [10]. Furthermore,
the improvement on 13×13 and 19×19 boards is worth being under-
lined: 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 5h to play one 19×19 game,
while Indigo plays out a full 19×19 game in a couple of minutes.

4.5. Olga\((pseudo = true, \text{ preprocess} = true) vs its author\)

Fig. 1 shows a game between Olga \((pseudo = true, \text{ preprocess} = true)\) play-
ing 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 pro-
grams: 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

| Table 5
| Results of Indigo2002 and Olga\((pseudo = true, \text{ preprocess} = true, \ N_s = 10, \ N_m = 100, \ r_d = 2.0, \ \sigma_e = 0.4)\) against GNU Go-3.2 |
| Board size | Indigo2002 | Olga\((true, true)\) |
| Board size | 9×9 | 13×13 | 19×19 | 9×9 | 13×13 | 19×19 |
| % wins | 35% | 13% | 6% | 37% | 33% | 19% |
| Mean | −9 | −34 | −83 | −5 | −15 | −40 |
| Time | 20’ | 1’ | 2’ | 12’ | 1h | 5h |
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.

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.

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 $19 \times 19$ boards and under reasonable time constraints (1 h 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 $19 \times 19$ competition [13] and 4th upon 10 programs in the $9 \times 9$ 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 $3 \times 3$ 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 $9 \times 9$ 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