Learning and Problem Solving in Gogol, a Go playing program

Tristan CAZENAVE
LAFORIA
Université Pierre et Marie Curie
Place Jussieu
Paris Cedex 05
FRANCE
e-mail : cazenave@laforia.ibp.fr

Learning and Problem Solving in Gogol, a Go playing program

Abstract

Gogol is a system which learns to reach goals and solve problems specific to the game of Go. It understands its mistakes, and then modifies itself so as not to make them again. It can learn by playing against itself, against a human or computer opponent, or when analyzing games played by other players. The learning process is not specific to the game of Go, it can be applied to other games.

Key words : machine learning,planning, application, Game of Go, Metaknowledge

Résumé

Gogol est un système qui apprend à atteindre des buts et à résoudre des problèmes spécifiques au jeu de Go. Il comprend ses erreurs, puis se modifie pour ne plus les commettre. Il peut apprendre en jouant contre lui-même, contre un adversaire humain ou informatique, ou en analysant des parties jouées par d'autres. Le processus d'apprentissage n'est pas spécifique au jeu de Go, il peut être appliqué à d'autres jeux.

Mots clés : Apprentissage, Planification, Jeu de Go, Métaconnaissance

Introduction

The game of Go is a game of strategy, but unlike chess or some other games well approached in Artificial Intelligence, no general algorithm has yet been found to provide computers with a good level of play. Traditional algorithms do not work due to the size of the board (19*19=361 intersections), leading to an average of 200 legal moves, and to the difficulty of evaluating a position which needs a lot of expertise. It is an ideal ground to test new techniques for learning to play games. Today's programs are based on expert systems (Kierulf 1990), but they are limited by the maintenance of their large bases of rules. Experts have problems introducing some new knowledge without causing unpredictable interferences with prior knowledge. A learning approach which automates the discovery and the integration of some new knowledge seems attractive (Pitrat 1990). That is the approach used in Gogol, a learning program to play Go.

Representation of Knowledge

Games

I use Conway's games theory (Berlekamp 1992) to represent Go knowledge. A game is associated with a goal, it can have different states :

- if the computer can reach the goal whatever the opponent tries, the game is '>'.
- if the computer can reach the goal by playing first, but cannot reach it if the opponent plays first, the game is '*'.
- if the goal cannot be reached whatever the computer tries, the game is '<'.

Goals

Different goals recognized as important ones by Go players are implemented in Gogol, each goal being associated with a game. The main goals are the following :

- Making an eye, i.e. reaching one of the positions below :

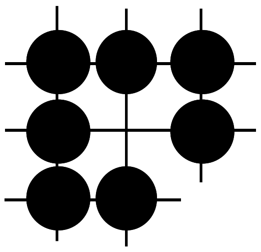


Figure 1 : Black eye

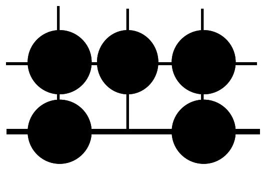


Figure 2 : Black eye on the edge

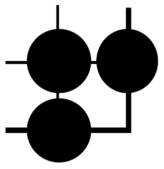


Figure 3 : Black eye in the corner

- Connecting two stones, making a chain of adjacent stones between the two stones.

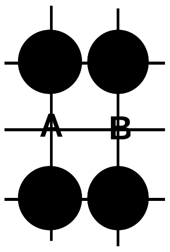


Figure 4 : Bamboo join

In this example, the game of connection is > for black. If white plays A, black plays B and is connected, and if white plays B, black plays A. A Go proverb says the bamboo join is unbreakable.

- Removing a stone from the board, there is a rule saying that every surrounded block of stones should be removed from the board. This is one of the goals the program tries to reach.

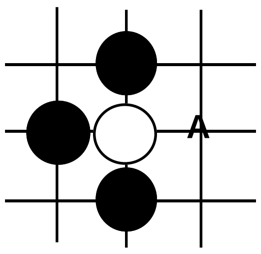


Figure 5 : the game of eating is * for the white stone

If black plays A, he eats the white stone, but if white plays A, black is no more able to eat the stone, so the game of eating the white stone is *.

Concepts

Many different concepts are used as premises of the rules, they are all concerned with liberties and the type of the block of stones to which the rule applies. For each target block, for example the block the program wants to eat, it determines the adjacent blocks and the adjacent blocks to the adjacent blocks. The information about the nature of the block will be used as a premise of the rule.

Two stones are in the same block if they can be linked by adjacent stones of the same color. A liberty is an empty intersection near a stone.

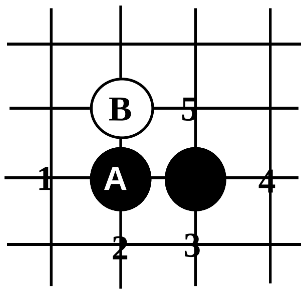


Figure 6 : Block A has 5 liberties and two stones, block B has 3 liberties.

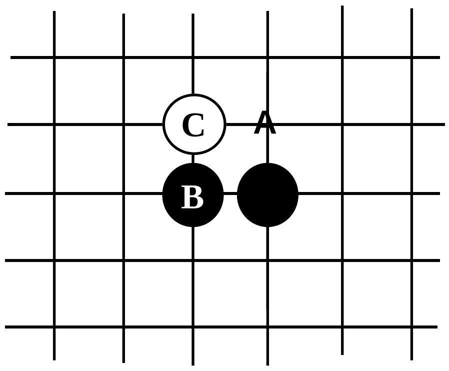
For each stone, the program can also test :

- the number of liberties of a stone.
- the number of liberties of all the adjacent blocks .
- the number of liberties of the target block.
- the number of stones in a block.

For each empty intersection, it can test :

- the number of liberties it gets if it plays there.
- the number of liberties the opponent gets if he plays there.

Values of different concepts on an example are given in the figure below :



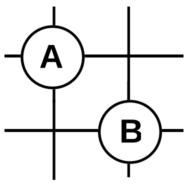
The target block is B, the adjacent block is C,

the number of stones of the target block is 2,
the number of liberties of the target block is 5,
the number of liberties Black gets if he plays A is 6,
the number of liberties White gets if he plays A is 4.

Figure 7 : Examples of some concepts

Rules

There are two types of rules : static rules and dynamic rules. A static rule is the association of a pattern (a part of the goban) with a conclusion about a game as shown in figure 8. A dynamic rule is the association of a pattern with information on the liberties of different stones as well as statistics on the results of the rule, concluding on a move to try in the Alpha-Beta tree.



Conclusion : connect A and B
Game : >

Figure 8 : Static rule

Problem Solving

To have a good level of play, a Go program has to solve problems. One practical reason for this is that humans can solve Go problems and that they are better than programs. Another one is that programs with problem solving abilities play better than programs without such abilities. The third reason is that, in order to know whether a stone is taken in shisho (see figure 9) which is a subpart of the game of eating a stone, you need two rules and little calculation if you accept to calculate, but you need thousands of static rules if you don't want to calculate. For these three reasons, Gogol is able to solve problems by using rules saying what moves to try and what moves not to try.

Controlling trees

To avoid combinatorial explosion, Gogol considers only a few moves at each level of the Alpha-Beta tree. It permits to go down to a level deeper than 40, giving correct results much of the time. To do so, it has to determine among all the moves the ones that have the best chance of winning, or those that are the most necessary not to lose.

Cutting trees is unavoidable. In Go it often happen that a very simple look ahead goes deeper than 20 moves (especially in the case of shishos). But a Go player only considers one move at each level of the tree. Gogol tries to do the same things as Go players, considering only one or two moves at each level of the tree.

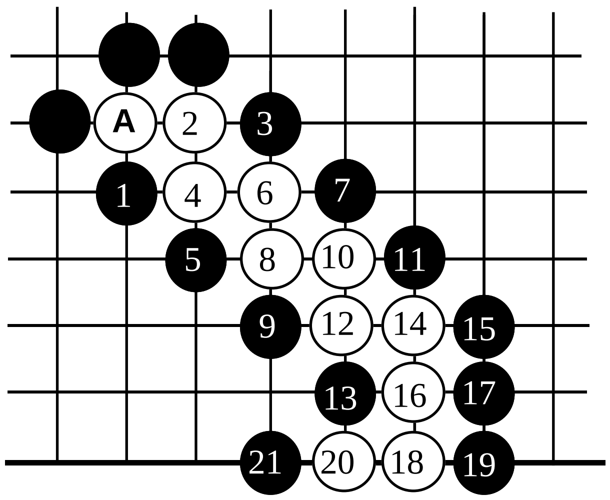


Figure 9 : A shisho (stairs in Japanese)

In the example of figure 9, Gogol only tries the marked moves as a human does. Even beginners are able to read shishos. Its goal oriented research and its rules for drastically cutting the tree enables to know the good result in 21 nodes, whereas a traditional minimax algorithm, as there is an average of 200 legal moves per position, would need 200^21=2.1*10^48 nodes to get the same result. A computer evaluating 1000 positions per second would approximately take 6.6*10^37 years. Needless to say it is not a reasonable time for solving a small simple sub problem of the game of Go.

Move ordering

Gogol maintains statistics on two properties for each rule. The first one is statistics to approximate the probability of winning if the rule is applied. The second one, called urgency of a move, is statistics to approximate the probability of losing if the rule is not applied. The urgency of a move has proved to be a better indicator of which move to consider first. The probability of winning is also used for choosing among disjunctive subgoals.

I am going to explain why the urgency of a move is a better indicator than the probability of winning :

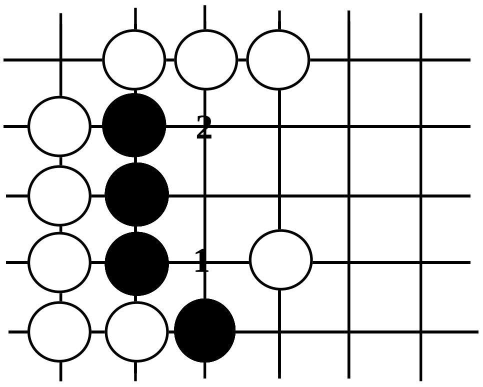
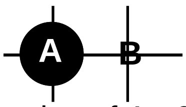


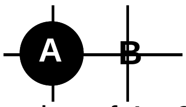
Figure 10 : 1 or 2 to eat the black stones?

Rule advising 1 :



Number of liberties of A : 3
Number of liberties the opponent gets if he plays B : 6
Type of A : Block
Goal : eat A
Move : B
Chance of winning : 25 %
Urgency of move : 100 %

Rule advising 2 :



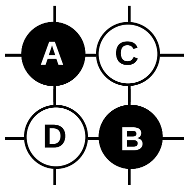
Number of liberties of A : 3
Type of A : Block
Goal : eat A
Move : B
Chance of winning : 60 %
Urgency of move : 30 %

The correct move is 1, it is chosen if we prefer rules which have a high rated urgency rather than rules which have a high rated chance of winning.

Goals and subgoals

In this section, I will show the necessity of having subgoals associated with a principal goal, and how the interaction of subgoals should be treated.

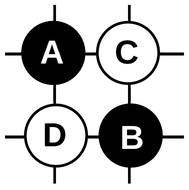
A way to connect two stones separated by an opponent stone is to eat the opponent stone. So eating is a subgoal of connecting. This separation between goal and subgoal is necessary, because a move for eating a stone can be on the opposite side of the board. To see if a stone can be eaten, Gogol examines all the moves related to the block of the stone, the adjacent blocks and the adjacent blocks to adjacent blocks. If there is only one stone to be eaten, the choice between the moves is obvious, Gogol takes the most urgent move. A problem arises when it has to choose between eating two different blocks as in Figure 12.



Goal	: Connect A and B
Subgoal	: (eat C) OR (eat D)

Figure 11 : Subgoals disjunction

In this case, Gogol should no more choose the most urgent move, because of the disjunction, it should choose the move which has the biggest chance of eating one of the two blocks. To do this, it determines the block which has the highest probability of been eaten, and then chooses the most urgent eating move for this block. So it takes the maximum urgency of the minimum probability of winning.



Goal	: Disconnect A and B
Subgoal	: (save C) AND (save D)

Figure 12 : Subgoals conjunction

When there is subgoals conjunction, Gogol chooses the most urgent move, mixing the moves for eating the two blocks. It takes the maximum urgency upon the two blocks.

Learning

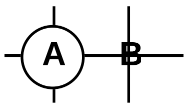
Very interesting researches have been carried out on learning chess (Pitrat 1974), but learning programs are not as good as combinatorial programs. I am particularly interested in understanding how humans learn, for I think it is the best way to construct a good Go playing program as blind search does not work for Go. A good Go program needs a lot of rules, as well as some metarules to learn and use those rules. Experiments have been done on learning to play Go (Pell 1991, Stoutamire 1991) but they were all trying to learn the best move for the overall board without understanding the reasons why it was a good move. My approach is based on learning to solve subproblems of the game of Go. It enables a program to understand better what it is learning and why .
In this chapter I will show how Gogol learns with an example. This learning process can be

split into five parts.

Metarules to determine when learning should occur

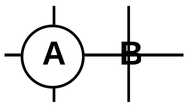
Before learning has occurred, Gogol has two rules telling it how to eat and how to save a stone.

Rule to save a stone :



Number of liberties of A : 1
Type of A : Block
Goal : eat A
Move : B.

Rule to eat a stone :



Number of liberties of A : 2
Type of A : Block
Goal : save A
Move : B.

In the situation below :

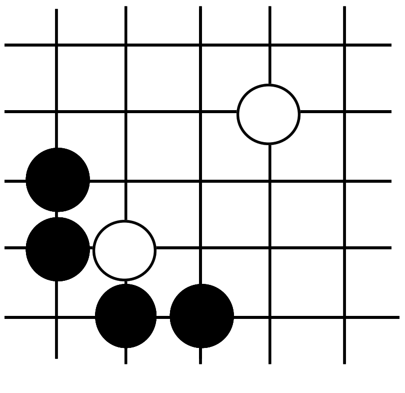


Figure 13 : Can the white stone be eaten ?

applying the rules to calculate if the stone can be eaten, Gogol tries the following moves:

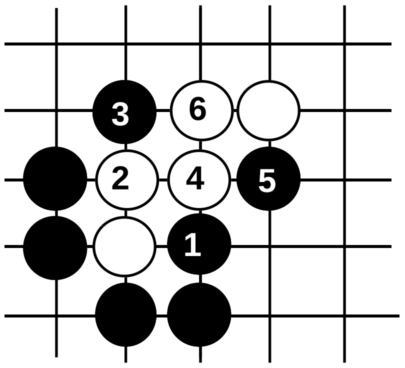


Figure 14 : Failing to take the white stone

After move 6, no rules permit to continue. So according to its knowledge, it cannot take the white stone. Hence, the game of eating the white stone is <.

But while playing, or while analyzing a game, it encounters the move below :

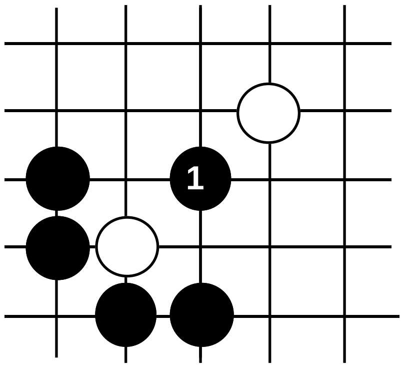


Figure 15 : A surprising move

It has a surprise when recalculating the game of eating the white stone :

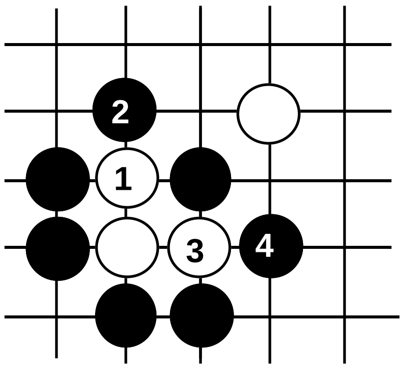


Figure 16 : The white stone cannot escape

Now the white stone cannot escape, though it believed it could before the last move. The reason for this surprise is that it did not try the black move when calculating the game. After this surprising move, it knows something is wrong with its rules, and it needs a new rule to try the unforeseen move next time.

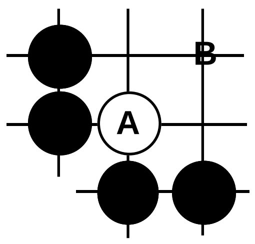
Another metarule which determines when learning should occur, applies when Gogol sees a rule is applied, but not for good reasons. In this case it creates new rules from the example. The rule permitting such a learning says :

> If a rule has given a working move.
> If this rule has not been applied first because of a lower urgency.
> If I am confident in the notation of the rule (more than 50 cases encountered).
> Then create new rules advising the working move.

Metarule for creating new rules

To create rules so as not to be surprised the next time a same situation arises, Gogol uses its understanding of the surprising move. It uses the intersections and blocks implied in the calculation of the sequence after the move to create patterns. It also uses the concepts presented above to create new rules.

In the previous example, it creates the following rule :



Number of liberties of A : 2
Type of A : Block
Goal : eat A
Move : B.

Metarules for modifying rules

Each rule is associated with statistics. After each tree calculation, it refreshes the statistics. These statistics are used for move ordering in the tree.
Another way to modify rules is to generalize a good rule, deleting an element of the pattern, decrementing a minimum value or incrementing a maximum value of liberties or stones.

Forgetting

The dynamic calculation of statistics during the learning process is a way of adapting the importance of rules to the new knowledge of the program. But to be as near as possible to the good value, Gogol needs to forget the statistics made with a wrong knowledge. To accomplish that, there is a mechanism which takes only into account the most recent

statistics.
Another way of forgetting is the deletion of rules. This happens when a rule is less general than another one and has worse statistics (provided they are relevant, i.e. made from a sufficient number of cases). It also happens when a rule has statistics of winning close to zero or when a static rule has proved to be false due to new moves tried in its calculation.

Learning static rules

Given a goal, Gogol generates all the possible patterns smaller than a 4x3 rectangle. For all the patterns, he calculates two Alpha-Beta, one with white playing first, the other with black playing first. It gives him the result of the game associated with the goal. It creates a static rule. Another module is charged when all the rules have been calculated to generalize them, replacing when possible "empty, black or white" points by "don't know" points.
Another way of creating static rules is to remember all the intersections implied in the tree when it is developed in a game. If all the intersections contain in a 8*8 square, the program stores the pattern with the result found.

Performances

Gogol has played many games against human opponents, most on 9x9 boards. Its level is about 17 Kyu. However, as a program it has a good level. It beats GnuGo, the best available domain public Go program, with a large advance (271 points on a 19x19 board, the maximum possible score being 361 points). It is a little behind the best commercially available programs, when playing a game against Many Faces of Go (Fotland 1993) or against Star of Poland on a 19x19 board, it is 10 to 60 points behind at the end of the game. The superiority of Gogol comes from its ability to learn from its mistakes, so each game it looses against a better program permits it to progress. It gets better every game.

To show the improvements of the program, I made it play regularly against a non evolving program. The progression of the results is shown in figure 17. This progression is due both to the learning mechanisms described in this article and to the addition of strategic knowledge.

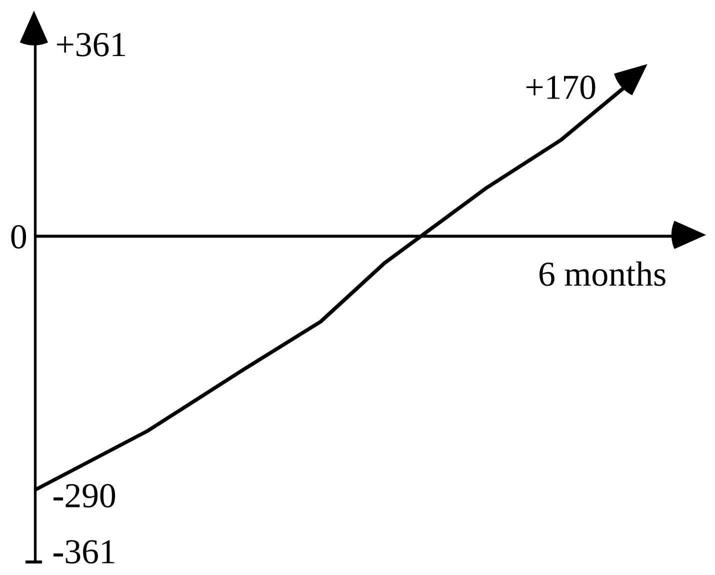


Figure 17 : Progression against another program

Gogol is written in C++, it has 25000 lines of code and 20 classes, it plays a move on a 19x19 board in less than one minute on a Sparcstation 10.
It has learned more than 9000 static rules but does not use all of them. It has also learned about 200 dynamic rules, but this number varies when applying metarules for learning after it has played a game .

Conclusion

Computer Go, in spite of a lot of efforts due to different researchers, is still at a low level considering human performances.

One originality of Gogol is that it is evaluating moves as well as positions. It is even learning to evaluate moves.

Formalizing knowledge in a learnable way forces to clarify points which could otherwise have been unseen. Another advantage is the aptitude for modifying the program. But the main advantage remains the automatisation of the modifications which allows the program to progress alone, provided it can play or analyze new games.

Gogol understands its failures and uses this understanding to improve itself by learning new rules. Almost all the rules it uses are rules it has learned by itself.

Bibliography

Berlekamp 1982 E. Berlekamp, J.H. Conway, R.K. Guy. Winning Ways.
Academic Press, London 1982.

Bradley 1979 Bradley, M.B. The Game of Go - The Ultimate Programming Challenge ?
Creative Computing 5, 3 (Mars 1979), 89-99.

Bouzy 1994 Bruno Bouzy. Modélisation des groupes au jeu de Go. Laforia, 1994.

Cazenave 1994 T. Cazenave. Système Apprenant à jouer au Go.
2ième Rencontres Nationales des Jeunes Chercheurs en Intelligence Artificielle, Marseille 1994.

Fotland 1993 David Fotland. Knowledge Representation in the Many Faces of Go.
Second Cannes/Sophia-Antipolis Go research day, February 1993.

Hsu 1990 Feng-Hsiung Hsu, Thomas Anantharaman, Murray Campbell, Andreas Nowatzyk. Un ordinateur parmi les grands maîtres d'échecs. Pour la
Science
n° 156, Octobre 1990.

Kierulf 1990 Ander Kierulf, Ken Chen, Jurg Nievergelt.
Smart Game Board and Go Explorer : A case study in software and knowledge engineering.
Communications of the ACM, 33 (2), February 1990.

Lee 1988 Kai-Fu Lee and Sanjoy Mahajan. A pattern classification approach to evaluation function learning. *Artificial Intelligence*, 36:1-25, 1988.

Pell 1991 Barney Pell. Exploratory Learning in the Game of Go.
In D.N.L. Levy and D.F. Beal, editors, Heuristic Programming in Artificial Intelligence 2 - The Second Computer Olympiad. Ellis Horwood, 1991.

Pitrat 1974 Jacques Pitrat.
Realization of a program learning to find combination in chess.
Computer orientated learning processes.
NATO Advanced Study Institute.

Pitrat 1990 Jacques Pitrat. Métaconnaissances futur de l'intelligence artificielle.
Hermès 1990.

Stoutamire 1992 David Stoutamire. Machine Learning, Game Play, and Go.
Postscript file available by ftp ftp.pasteur.fr, /pub/Go/comp/report.ps.Z

List of Figures