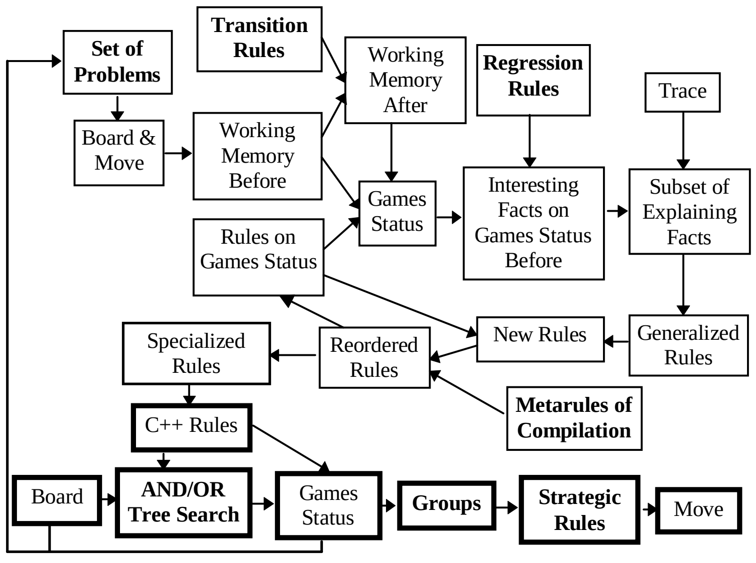
Tristan Cazenave
LIP6, Université Pierre et Marie Curie
4, place Jussieu
75252 PARIS CEDEX 05, FRANCE
Tristan.Cazenave@lip6.fr

France

Gogol
(An Analytic Learning Program)

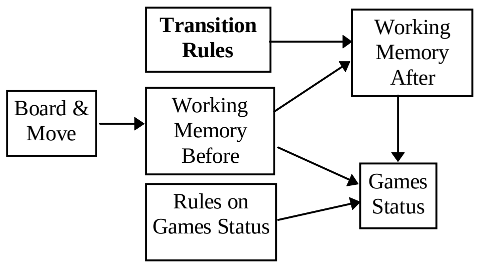
General architecture



→ = Is used to create

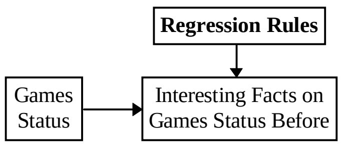
Bold characters indicate that the corresponding module has partly been written by a human. Bold rectangles represent the Go playing program. Normal characters represent the parts of the system that are created by the system itself. At the beginning of the learning process, the system is only given three sets of rules. The transition rules describe the direct effects of a move on the facts representing a board. Ten production rules define the six interesting subgoals of the game of Go: Connect-Disconnect, Take-Live and Make_an_eye-Remove_an_eye. The metarules of compilation enable to transform learned rules in order to use them efficiently. Using a set of problems, it creates a C++ program that is used to develop proof trees for the Go playing program. The creation of the C++ program follows a six step process described in the six following subsections.

Problem Solving



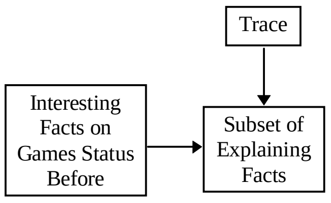
We use a deductive problem solving method. The system begins with transforming a board and its associated move into a *working memory before* containing approximately thousands of facts. Then it use the *transition rules* to deduce the *working memory representing the board after the move*. After that, it deduces the *games status* before and after the move, using the *rules on the games status*. Before learning, the system has only ten rules defining won game status.

Selection of interesting facts



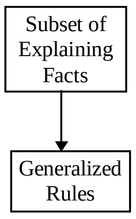
To select interesting facts, the system compares the game status before the move and after the move. If the game status after the move is more accurate than the game status deduced before the move, then the fact describing the new game status is interesting to explain so as to create a new rule which will enable to deduce it the next time.

Self-Observation


Trace
Interesting Facts on Games Status Before
Subset of Explaining Facts

Once it has selected some interesting facts on game status, the system uses the trace created while solving the problem to explain why the interesting facts have been deduced. The result of this explanation is a subset of the facts contained in the working memory before the move.

Generalization



The rule obtained after self-observation is composed of instantiated variables and constants. This rule is very specific to the problem solved. So as to use it in many other cases, it is generalized by replacing instantiated variables with variables. This generalization procedure is truth preserving. The rules created this way are theorems of the learned game. They always give a true conclusion. On the contrary of previous approaches, we never over-generalize rules. We approach the optimum as shown in figure 2 and not like in figure 1, which is the classical approach. This property is very important when learning to develop proof trees. Otherwise, the learned rules are not theorems and cannot be used to develop proof trees. Moreover, we do not want to learn false rules because this would lead to learning more and more false rules.

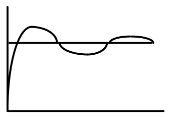


Figure 1

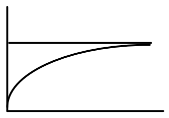
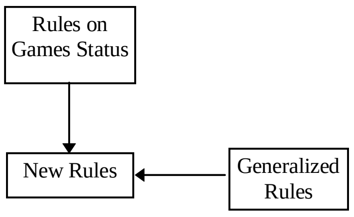


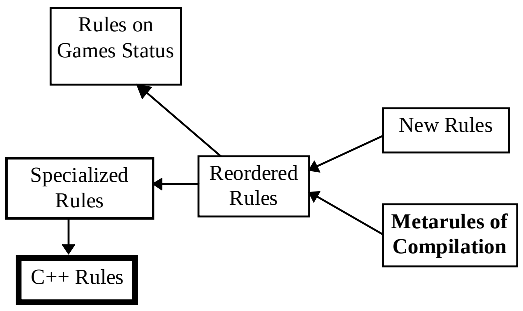
Figure 2

Adding and removing rules



The system removes rules which are too specific by unifying generalized rules and rules previously learned. This way, only new rules are included in the system, and too specific rules are removed.

Compilation



Reordering conditions
A good ordering of conditions can provide big speedups in production systems. To reorder conditions in our learned rules, we use a very simple and efficient algorithm. It is based on the estimated number of following nodes that the firing of a condition will create in the semi-unification tree.

Compiling in C++
Another source of inefficiency is the interpretation of production rules. When an interpreted problem solver instantiates a variable, it has to traverse trees representing the working memory, to create a linked list of the instantiations of the variable and to go through this linked list. Instantiating a variable or making a test requires many instructions at the assembly language level. If a rule is compiled into a C++ program, tests are represented by only one instruction and multiple instantiations by a simple loop.

Strong point of Gogol: Analytic Learning

Gogol learns to play Go using the rules of the game and the definitions of some interesting goals. It specializes the goals using the rules of the game. The resulting rules are compiled into a C++ program. The system which has learned the tactical Go rules (Introspect) is general and the learning algorithm has been used in other domains.

As it is impossible to search the entire tree for the game of Go, the best Go playing programs rely on a knowledge intensive approach. They are generally divided in two modules :
- a tactical module that develops narrow and deep search trees. Each tree is related to the achievement of a subgoal of the game of Go.

- a strategic module which chooses the move to play according to the results of the tactical module.
The tactical module usually uses patterns to choose a few number of moves in order to develop the search tree. These patterns are numerous and hand-coded. Creating this large number of tactical patterns requires a high level of expertise, a lot of time and a long process of trial and errors. Moreover, even the people who are expert in Go and in programming find it difficult to design these patterns. This phenomenon can be explained by the high level of specialization of these patterns: once the expert has acquired them, they become unconscious and it is hard and painful for the expert to explain why he has chosen to consider a move rather than another one. However, despite its great interest for Artificial Intelligence research, the best Go programs are not well described in the Artificial Intelligence literature. A Go program contains thousands of specific expert rules. Thus, it is difficult to describe them in a synthetic way.
The difficulty of encoding Go knowledge is the consequence of a well known difficulty of expert system development: the knowledge engineering bottleneck. The goal of Gogol is to avoid this bottleneck by replacing the knowledge extraction process with an automated construction of knowledge based on examples of problem solving. Machine learning techniques enable Go programmers to get rid of the painful expert knowledge acquisition. Thus, computer Go is an ideal domain to test the efficiency of the various machine learning techniques.

Previous Computer Go Tournaments

1996 Fost Cup : 12th out of 19, 4 wins, 5 losses**.**

Games Lost against :	Mutsuki, Jimmy, Go Intellect, Explorer and Takuchan.
Games Won against :	Katsunari, Tokyo96, Gooter, Goro.

How to get my program

Tristan Cazenave
LIP6, Université Pierre et Marie Curie
4, place Jussieu
75252 PARIS CEDEX 05, FRANCE
Tristan.Cazenave@lip6.fr
http://www-laforia.ibp.fr/~cazenave/Tristan.html