Go Review
reprinted from: Proc. IFIP Congress 62 pp 192-194

# SIMULATION OF A LEARNING MACHINE FOR PLAYING GO

H. REMUS

IBM Laboratories, Boeblingen / Wuerttemberg, Germany

## 1. Introduction

This report describes a program which supplies a minimum of information on the game GO to the computer. It is an extension of ideas which have already been described[1]). For the program, the rules of the game are given, as well as parameters connected with a possible strategy, and directions for evaluating successful and unsuccessful moves for future behaviour. The requirements for the program, and the resulting analogies to learning processes are discussed.

GO is especially suited to these experiments, because the rules are simple, no mathematical theory of the game is known, and it is impossible to calculate all the variants because of the multiplicity of courses a game may take.

## 2. Rules of GO

GO is a two-person game played on a board[2,3]). Black and white *pieces* are set alternately on a 19×19 board and there is no limit to the number of pieces which may be set. Horizontal and vertical rows of pieces of the same colour form *chains* (fig. 1). A vacant point contiguous to a chain in the horizontal or vertical direction is a *degree of freedom*. A chain (or a single piece) without any degrees of freedom is a *prisoner;* it is removed by the opponent and its pieces are counted up at the end of the game. Making moves or taking prisoners is not compulsory. *Suicide* is not allowed, i.e. a player may not make a move which will deprive one of his own chains of all degrees of freedom (a move is regarded as complete only after any prisoners have been removed). Areas in which the opponent is not allowed to set, or because of disadvantages does not wish to set, are *taken areas*. Final scoring at the end of a game and the decision about loss or gain is made by adding up the prisoners and the points in the taken areas; an *area* being a chain of vacant points surrounded only by pieces of the same colour. There may be areas which are surrounded by pieces of either colour and in which neither of the players may set without disadvantage (Seki). These areas are not scored. A game is finished when neither of the players is able or wishes to continue.

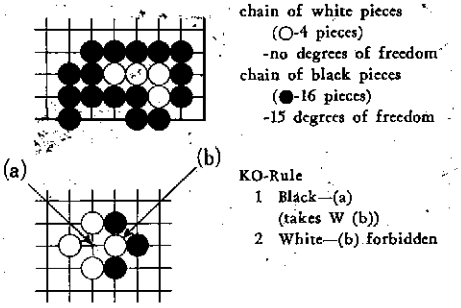


Fig. 1. Definitions of Go.

There is one special rule, the so-called *Ko-rule* which forbids a player who has lost a single piece to take prisoner, in the subsequent move, just the one piece by which it was taken (this prevents a board position being repeated indefinitely).

The 19×19 board permits $3^{361} \sim 10^{172}$ different board positions in the course of a game.

## 3. Knowing Phase of the Machine

For the present program, the board is

reduced to 11×11 giving $3^{121}\sim10^{57}$ possible board positions.

Each position is described by three subsets (sets of points occupied by black pieces, and by white pieces, and the set of empty points). The subsets are represented by 11×11 matrices, $A_1$, $A_2$, $A_3$; where each matrix element corresponds to a point on the board. If the point belongs to the respective set, the matrix element is 1, otherwise it will be 0.

By this separation of positions, and by employing disjunction, conjunction, and negation, as well as by introducing shifts and an instruction for counting the ones in a matrix, the rules of the game and the formulae for the final scoring can be clearly fixed.

During the game (fig. 2), the main program determines the permitted moves, using the rules of the game. First of all, the lexicon is searched to see whether a formerly successful move is known for the current position. The lexicon contains position classes for which the same move was successful. An operator results, which filters the number of possible moves given by the main program, retaining those which show a maximum ratio of success. (If the position cannot be found in the lexicon, the intial number of permitted moves remains unchanged).

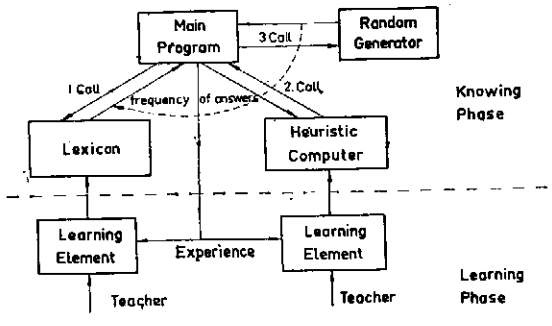


Fig. 2. Structure of the Go-machine

If, according to the lexicon, more than one move is possible, it is left to the heuristic computer to decide which move should be made. For 16 parameters related to the game, the values for any possible move are calculated (e.g. the real and possible degree of freedom of chains, number of prisoners taken, distance from the last piece set by the opponent, number of own and opponent's pieces in certain areas), and the corresponding success weightings are looked up in a list. A linear combination of these success weightings produces a decision-making criterion. The latter determines the most favourable move, taking into account the subsequent six half moves, by applying a minimax procedure. If more than one move is possible, then the random generator selects the move to be made.

Analogous to learning processes, the moves to be made in the initial stage will be determined mainly by the random generator, but subsequently and to an increasing extent, they will be determined by the heuristic computer, and eventually by the lexicon.

## 4. Success Weighting

The rule for the final scoring is taken as the criterion for the success weighting of each move, i.e. in each position it is assumed that the game is terminated acknowledging only those areas which are left at the end of the game. Taking the values of the move under review, as well as the subsequent five half moves (with the weights 6, 5, 4, 3, 2, 1), a success curve is produced. (Thereby, the weighted value of the last move is introduced for the five subsequent imaginary half moves which are necessary for a calculation.)

The moves are divided into three groups. All moves for which the success curve increases are considered *good*, all moves of the loser for which the success curve decreases more rapidly than in the preceding move are considered *bad*, and the corresponding moves of the winner as *neutral*. All other moves are considered *good* for the

winner and *neutral* for the loser. If a game ends in a draw, both players are supposed to be winners. This separation of the success weighting into subgoals shows a certain relationship to other learning programs[4]).

The above-mentioned weighting was used for the test game in[3]), played on an 11×11 board. The game had 126 moves, and white won. The following combinations of moves resulted:

| | + | = | − |
|---|---|---|---|
| White | 61 | 2 | — |
| Black | 45 | 16 | 2 |

In practice, only the 55th move (black) did not correspond to the explanations. The commentator called this move a *good* one while it was shown as *neutral* by the success curve.

## 5. Learning Phase of the Machine

After completion of a game the latest experiences are evaluated. Due to the interrelation of the success weighting of a move and the final result of the game, it is necessary to separate the program into *learning* and *knowing* phases. It is quite possible that a weighting method could be found which would not depend on loss or gain; thus obviating this separation.

The moves considered *good* are registered in the lexicon; if the successful move is identical with a move of one of the existing classes, and if 23 or more points coincide in the 5×5 vicinity of the move, position $A$ extends class $K$ as follows:

$K_r^{(n+1)} = K_r^{(n)} \cup A_r$, for $r = 1, 2, 3$.

Thus for the three subsets of a class, the points have the following meanings for concurrence with points of positions:

111 may be occupied
110 must be occupied
100 must be occupied by black
010 must be occupied by white
101 must not be occupied by white
011 must not be occupied by black
001 must not be occupied

Classes are considered equivalent, with regard to possible rotations and reflections, as well as with regard to permitted shifts (points with 111 may be shifted cyclically over the edge). If there is no lexicon class in which the respective position may be registered, a new class is established.

If *bad* moves occur in a lexicon class they are eliminated, together with the corresponding board positions, as follows:

$$K^{(n+1)} = K_r^{(n)} \cap (\overline{A}_r \cup (\overline{K}_s^{(n)} \cap \overline{K}_t^{(n)}))$$

for the three subsets with different $r, s, t$.

The parameters of the heuristic are calculated for all moves. If a move is considered *good*, the success quotients $a/b$ of the lexicon class and the table for the heuristic computer are changed to $(a+1)/(b+1)$. If the move is *neutral*, only the divisors of the success quotients of the heuristic computer are raised by one. If the move is *bad*, the success quotients of the affected lexicon classes and the heuristic table become $a/(b+1)$.

After the learning phase has been terminated, the lexicon is reorganized and all classes with a success ratio smaller than 0.05 are eliminated. Care is taken that equivalent classes are represented only once. The number of remaining classes is restricted to 1000; if there are more than 1000 classes, those with the lowest rate of applicability are removed. Afterwards, the classes are resorted to reduce the duration of calls during the next game.

The machine learns not only by its own moves but also by those of its opponent.

## 6. Programming and Examples

The IBM 704 computer with a total of 32768 36-bit words was chosen for the simulation. At the date of this report, programming was nearly completed. The programs are written in SAP language (Symbolic Assembly Program). There are about 8000 instructions, constants and general intermediate storage (including

input and output subroutines).

15,000 storage positions are provided for the lexicon. Moreover, the table for the heuristic computer requires 672 words, and the minimax procedure 600 words. The lexicon is sorted by means of two magnetic tapes.

The following formula serves as a random number generator:

$$127(j+a_i)-d\,2^{35}=f\,a_{i+1}+b,$$

where $j$ is the number of the move to be made next, $a_0$ an input constant (an integer greater than 0), $d$ a positive correction integer to keep the left-hand side of the equation smaller than $2^{35}$ (and greater than 0), $f$ the number of possible moves, and $b$ the random number to be used. (For $b$ the following possibilities exist; 0, 1 . . . , $f$—2, $f$—1.)

It takes about 100 msec to calculate the decision factor for one move. The duration of a total investigation of a lexicon class for analogy to a position with possible shifts, rotations and reflections is estimated to be 0.5 sec.

Some examples were simulated to test the program. In the following results the performance of the heuristic computer in formerly non-existing positions is shown. As machine input the test game from [3]) was chosen.

In Example 1 (fig. 3a) the machine playing white, finds among the three permitted moves a1, b1 and a3, the move b1. This is obviously the most effective of the three, for in the case of black answering with a1 white follows with a2, and the three black pieces are taken. If white had chosen one of the two other moves, black could have prevented the loss.

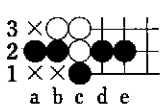


Fig.3 a.

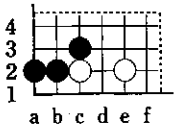


Fig.3 b.

In Example 2 (fig. 3b), with its last move c3 black has attacked the white piece on c2. Of the 19 possible moves in the area a-f, 1-4, white selects the move e1. This is not as secure as d2 but white is able to save c2 by following black's answer of d2, with c1.

Before being familiar with the test game, the machine would of course, have reacted with random moves.

## 7. Final Remarks

The main object of the initial investigations with a completely tested GO program should be to answer the following questions:

1. What is the course of development of the lexicon? Does the above-mentioned method suffice for forming classes to *mark* a maximum number of favourable situations and moves?
2. How is the weighting table influenced by experience? i.e. Do really significant differences emerge in the elements of this table, or do functional relationships result? It is quite possbile that some alterations may be made with regard to the selection of parameters.
3. Does the method described for numerical success weighting correspond in general to the human analysis of a game? This question affects the first two points, i.e. an insufficient separation into subgoals would mean—from the start—a negative answer to questions 1 and 2.

It is almost certain that a repetition of particular erroneous moves can be eliminated by means of statistical elements. Success criteria and rules for the utilization of experiences must be found.

The learning procedures under review are not only concerned with rote learning[5]); the recognition of the position characteristics in the lexicon classes, and the different significance of the employed parameters could be described as learning by means of comprehension[6]), following given directions, of course. The stored information changes

continuously; thus the learning process is reversible.

The program should be considered in terms of these analogies to learning processes and not of its ability to play GO.

## 8. References


1) Remus, H.: *Lernversuche an der IBM* 704. Lernende Automaten, Munich (1961) 144.
2) Takagawa, K.: "How to Play GO". Japan, 1956.
3) Dueball, F.: "Das Gospiel". Minden/Westf., 1960.
4) Newell, A., J. C. Shaw and H. A. Simon: *Report on a General Problem—Solving Program.* Information Processing, Paris (1959) 256.
5) Samuel, A. L.: *Programming Computers to Play Games.* Advances in Computers, New York (1960) 165.
6) Zemanek, H.: *Beschreibung von Lernvorgängen.* Lernende Automaten, Munich (1961) 9.


## ABSTRACTS


The rules of GO are formulated in mathematical language so that a program for playing the game can be developed. A precise determination of the best move in a given situation is not possible however, because of limitations of time and storage. By applying three operators successively—a lexicon, a heuristic computer and a random number generator —that move is selected which, according to present experience, is the most favourable of those permitted.

The lexicon and heuristic computer are built up and continuously improved by the machine itself, using the success or failure of previous moves. (The success of a move is deduced from the rules for the final scoring which are valid for each position). Thus, the effectiveness of these two operators increases in the course of time analogously to learning processes; i.e. during the initial games the moves will be determined mainly by the random number generator, but subsequently and to an increasing extent, by the weighting table of the heuristic computer, and eventually by the lexicon. With the aid of examples it is shown how far the experience gained will improve the quality of the game in formerly unencountered situations.


## DISCUSSION

H. C. Ratz (*Canada*). Do you know, or can you predict, how the performance of your machine compares with that of a human playing the same game?

H. Remus (*Germany*). I am not ready to predict this.

N. Blachman (*USA*). Does the program tell the computer when the game is over? If so, does it always do it correctly and does it count the score?

H. Remus (*Germany*). The machine does the final scoring. No method has yet been found to see the end of a game before all the areas are definitely taken.

N. Teufelhart (*Austria*). If two computers, supplied with the same information, play against each other, would you expect them to improve their knowledge of the game?

H. Remus (*Germany*). I expect them to increase their knowledge as long as random moves were involved.

W. Pavel (*Germany*). Is there any possibility of evaluating the initial moves? Is it possible to verify that the stars, the so-called "Hoshi" occupied in a handicap game, are optimal points? An interesting problem is the coherence between handicap and degree. If two opponents of equal degree play a handicap game what will be the outcome? The exact value of any handicap is not known. The difficulty in evaluating this statistically in games among men is that you cannot find two players of exactly equal degree, and also that *one* player will not be equal in degree in two different games. May it be possible to find the value of a handicap with the aid of such a program?

H. Remus (*Germany*). The machine may, of course, play a handicap game but there is as yet no intention of solving the question of optimal points. I realize that there are difficulties in teaching the heuristic computer the initial moves, but the lexicon will handle them.

M. Euwe (*Netherlands*). It seems to me that learning principles can mainly be applied to a game with uniform pieces and standard patterns and not to a game such as chess. Exactly the *same* position rarely occurs in chess, while *similar* positions may have very different possibilities. Even if it were possible to store the positions the machine had lost, it would not be easy to decide which was the critical move which caused the loss of the game.

H. Remus (*Germany*). I agree.