Mathematical modeling and analysis of learning techniques for the game of Go

Arturo Yee and Matías Alvarado

***Abstract*—**In this paper the mathematical modeling for Go game is by using Finite State Machine. The purpose in Go gaming is territorial control, and it is a hard challenge for computer sciences, since it conceals a huge combinatorial complexity that impacts the options for learning automation. In this paper, we show how the use of pattern recognition by neural networks (NNs) is quite efficient during the first and middle stage in a Go match, while Monte Carlo Tree Search (MTCS) during the match ending stage. The experimental results on the use of NNs and MCTS illustrate our claim, and precede the analytical comparison of major approaches based on NNs for Go automation.

***Keywords*—**Go Gaming, Tactics Pattern Recognition, Strategies Building, and Neural Network learning.

I. Introduction

THE formal analysis of the board game called "Go" is at the core of advances in computer science in the same way the analysis of Chess was during the 20th century [1]. Go is a top complex board game and currently, the deployment of learning algorithms for Go gaming automation is a central challenge for computational intelligence to demonstrate sufficient skill to beat the top human Go masters. The Go game official board (*goban*) is a 19 × 19 grid for two players using black-stones versus white-stones with zero-sum, deterministic, and perfect information [2]. By turn, each player places one black/white stone on one empty intersection or point of the board. Black plays first and white receives a compensation *komi*, by playing the second turn [3]. The goal of Go gaming is to control as much of the board area as possible, by means of complex strategies, applied through simple Go rules. Fig. 1 presents the flow diagram for Go gaming.

Arturo Yee is Ph. D. students in Computer Science Department at CINVESTAV-IPN. México City, México. Phone +52 55 5747 3756 Ext. 6555; e-mail: ayee@computacion.cs.cinvestav.mx.

Matías Alvarado is a research scientist in Computer Science Department at CINVESTAV-IPN. México City, México. Phone +52 55 5747 3756 Ext. 6555, e-mail: matias@cs.cinvestav.mx.

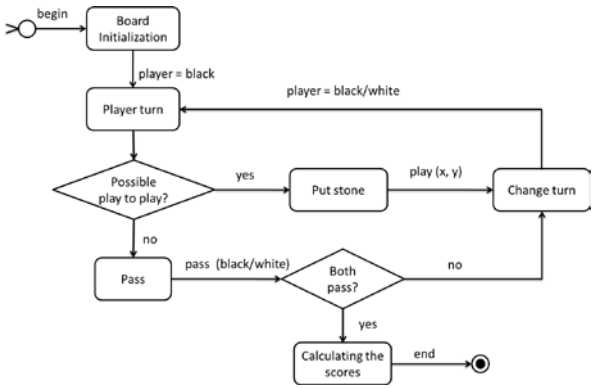

Fig. 1 Go gaming flow diagram

To determine the available moves during an automated Go match is a tough problem because of the huge search space to assess. The complexity of computer Go gaming is measured by the game tree size and the state space is quite descriptive. The game tree is the cardinality of the set of all possible manners for a legal sequence of moves, through all Go matches. A Go gaming state (node) is a particular board arrangement, i.e., the positions of the stones on the board at a specific moment in a match. The size of the Go game tree is around 10360: 10172 for the state space and up to 361 legal moves [4]. Therefore, the search space on Go gaming solutions is huger –very much– than that for Chess [2]. The moves of a Go match are depicted graphically by a decision tree that records the moves and is an element of the game tree. The root state is the match beginning. Any node children are those positions reachable in one move.

In this paper, we quantify the efficiency of pattern recognition by neural networks (NNs) during an automated Go match. The best pattern recognition of tactics and strategies for Go gaming is achieved during the first and middle steps of a match, as we concluded based on experimental results. In these steps, the a-priori-knowledge-based moves are very efficient for Go gaming. In a broader computer science perspective, the correct solution of Go gaming automation may enlighten solutions in diverse fields, such as complex network analysis, fractal formation, and bioinformatics. A mathematical modeling of the game of Go using Finite State Machine (FSM) is presented. On a broad perspective the formal modeling of games is relevant on engineering for computer process simulation [5], or in economy for price-based coordination on hierarchical systems [6], among a lot of other fields. Complementary, concepts form engineering like the many

valued quantum computation is applied for modeling the games' dynamics [7]; or concepts of economy like the modeling of the gross-domestic product is taken as an ordinary differential game of pursuit, or as a hereditary game [8]. The entire relationship on this issue is beyond the scope of the present paper, but given its relevance, it is outlined briefly in the Discussion section.

A. Go Gaming

On the Go board, the liberty of a stone is any vertical or horizontal unfilled point adjacent to the stone, which sometimes can be shared with other stones. Once a stone is placed on the board it can be removed only when it is captured, which happens if it is surrounded by adversarial stones and thus, losing all its liberties. Black stones capture white stones and vice versa. Two or more stones of the same color joined by horizontal or vertical points form a chain stone that cannot be divided; diagonally adjacent stones are not in a chain. From now on, the term stone refers to both single stones and chains, but explicit differentiation is made when required. Any stone is alive if it cannot be captured, and it is dead if it cannot avoid capture. When a player places a stone that will result in immediate capture, this is called suicide, which is not allowed. The game ends when both players pass on their turn. Then, the score is computed based on both territory occupied by the player on the board and the number of captured adversarial stones. The winner is whoever has the largest total.

The **basic tactics** of Go of eyes, ladders and nets are used to dominate a local area [9] (see Fig. 2). An *eye* is a single empty point enclosed by stones of the same color, which cannot be occupied by an adversary's stone owing to the suicide rule. Two eyes inside a stone make its capture very difficult. A stone having only one liberty is in *Atari*. A *ladder* results from a sequence of moves that forces an adversary's stone into atari. A *net* is a set of stones (not always a chain) that surrounds an adversary's stone such that it could eventually be captured [10]. All these *a-priori*-known patterns of Go basic tactics should be recognized for a fair Go game, and are used for training the NN in learning their recognition.

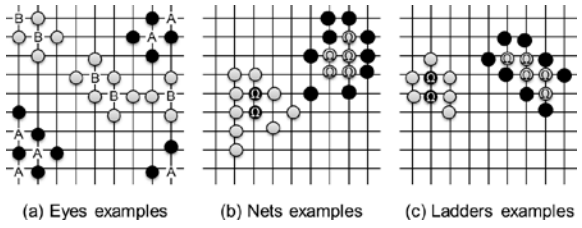

Fig. 2 Go basic tactics: (a) eyes, unavailable points – A is black and *B* is white; (b) Ω stones are surrounded by a net and may soon be captured; (c) Ω stones are in atari by ladders

For broad territory control, Go **strategies** follow a set of planned actions, deployed partly by using the aforementioned tactics as elements. Basic strategies are invasion, reduction, connection, and capture [9] (see Fig. 3). An *invasion* strategy places a stone near friendly stones, in an area where the adversary's stones look likely to dominate. A *reduction* strategy places a stone near friendly stones, to connect them if needed, in an area likely to be occupied eventually by the adversary. *Capture* reduces the liberties of an adversary's stone to zero and removes it from the board.

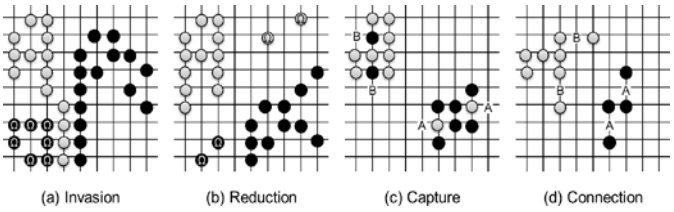

Fig. 3 Go basic strategies: (a) Ω stones perform invasion in territory dominated by white; (b) Ω black/white stones perform reduction in territory of white/black dominance; (c) black/white playing in positions *A/B* capture white/back stones; (d) black/white playing in positions *A/B* perform connections with friendly stone

A Go gaming strategy move, from the root node to the leaves nodes, is aiming to win a match efficiently. Despite the disarming simplicity of the Go rules, Go gaming conceals a huge combinatorial complexity [4, 11] (see Table I) and therefore, the big complexity is to set up an efficient strategy for playing Go**.** The **state space complexity** is the number of all the possible arrangements of the game board, which in a 19 × 19 board is about $3^{19\times19} \approx 10^{172.24}$ for Go, whereas it is 10^{50} for Chess and 10^{18} for checkers. The branching factor for Go ranges from 200–300 possible moves at each player's turn; for Chess, the range is 35–40 moves. The **game tree size** is the total number of different matches that can be played and for Go that is $\approx 10^{360}$ (chess $\approx 10^{123}$ and checkers $\approx 10^{54}$). Even on the 9 × 9 board size, the state space and the game tree size is astronomically large [12].

Table I. COMPLEXITY OF GO, CHESS, AND CHECKERS GAMES

Game	*Board size*	*State space*	*Game tree size*
Go	19 × 19	10^{172}	10^{360}
chess	8 × 8	10^{50}	10^{123}
Checkers	8 × 8	10^{18}	10^{54}

B. State of the Art

The gaming level of ongoing Go automated player is not great successful versus top human Go masters yet [13], contrasting with the achieved by chess automated players. Best Go gaming automation can beat middle level human Go players nowadays [14, 15]. However, the deployment of efficient Go gaming automation is being strengthened. To advance this effort, we quantify the performance of automated pattern recognition for Go basic tactics, such as eye and ladder, and based on the tactics recognition, we perform smart reasoning on Go basic strategies, such as reduction and invasion, both at the early and middle stages of a Go match.

The relevant advances by the Monte Carlo Tree Search (MCTS), applied to overcome the huge complexity of Go gaming, should be complemented in order to achieve victory

over top level Go masters. Methods focused on simulation-based search algorithms [14, 16, 17] behave very randomly in the early stages of a Go match and produce high search complexity in choosing the next Go moves, because of the huge set of free board positions at this step. In contrast, using pattern recognition and *a-priori*-known movements, sets the basis for efficient Go strategies/tactics gaming. The Go game is based on *long-term influence moves* [14]. Moves made in the beginning of the match affect the outcome of later moves and thus, this highlights the relevance on making the correct decisions early in the Go match. MCTS is particularly free from expert knowledge and from tactical-solving guidance [18], and the memory of previous games' moves is based on a huge number of simulations [14, 16] that is very costly in time during the early stages of a match. Furthermore, in specific situations, it prevents the identification of any correct move because of the lack of a tactical search; it needs too many simulations per move to achieve an appropriate gaming move [18, 19]. Hence, an *a-priori-knowledge*-based method for movement selection is appropriate at this step [20].

One very difficult task for Go automation is the evaluation of non-final positions for estimating the potential of occupied territory [3, 21, 22]. Prospective methods for programming Go gaming are being deployed in simulation-based search algorithms, heuristic searches, machine learning, and automated knowledge-based decision making [23]. The main challenge in Go gaming automation is to deal with the huge number of forms in the board, which must be classified prior to deciding on the next correct Go move. Thus, the automation of Go strategies is hugely complex. The process of tactics pattern recognition is essential in learning how to make a Go move [24]. For our quantitative analysis we use:

- A neural network for Go tactics pattern recognition and basic strategy construction.
- A MCTS for move automation in the end stages of a Go match.

The rest of this paper is organized as follows. Section II describes the mathematical modeling of the game of Go. Section III reviews the use of Neural Network and MCTS for the game of Go. Section IV presents pattern recognition for Go tactics and strategies. Section V provides the analysis of the techniques by stage and Neural Network Go players. Section VI is the Discussion, followed by the conclusions in Section VII.

II. Mathematical Formal Modeling for the Game of Go

A. Mathematical definition

The mathematical definitions of the Go game concepts follow.

- Board: $T = \{(x, y) | 1 \leq x, y \leq n\}$, $n = 19$ is the official size.
- Let $\pi \in T$, $\forall \pi \rho(\pi) \in \{e, b, w\}$, $\rho(\pi)$ be the event in a board point; initially all $\rho(\pi) = e$ an empty point in T, and b, w denotes black, white, $\Lambda = \{b, w\}$.
- $Ps = \{\pi | \pi \in T, \rho(\pi) \in \Lambda\} \neq \emptyset$, is the set of occupied board points.
- $Mh(\pi_0, \pi_1) = |\pi_0.x - \pi_1.x| + |\pi_0.y - \pi_1.y|$ is the Manhattan distance between $\pi_0, \pi_1 \in T$.
- Stone: $st = (\pi_0, \pi_1 \ldots \pi_n)$, $n \in \mathbb{N}$, $\rho(\pi_0) = \ldots = \rho(\pi_n)$, and $Mh(\pi_i, \pi_{i+1}) = 1$. A single stone is by $n = 1$, otherwise it is a chain stone.
- Set of stones: $St = \{st \subseteq Ps \mid st$ is a stone.
- Set of liberties for $st \in St$, $L(st) = \{\pi \in T \mid \rho(\pi) \in T - Ps\}$ and $\exists \pi$ in st with $Mh(\pi, \pi_0) = 1\}$.
- atari: $st \in St$ is in atari if and only if (iff) $|L(st)| = 1$.
- eye: $|\bigcap i = 1$ to n $L(st_i)| = 1$, $n = 2, 3, 4$.
- ladder: $ld \in St$ is a ladder on st iff st is the other color to ld and is in atari by ld.
- net: $nt \in St$ is a net on st iff $\forall b \in L(st)$, $\exists st_x$ in nt and π in st_x such that $Mh(b, \pi) = 1$ that is, any liberty of st is adjacent to a stone of the adversary.
- $P = \{p_1, p_2\}$, p_1 and p_2 the black and white player.
- $A = \{a_1, a_2 \ldots a_m\}$ is the set of actions.
- φ: $S \times A \rightarrow S$ is the transition function (relation).
- S is the state space.

The FSM for Go gaming algorithms follow, using the previous mathematical definitions.

A. Finite state machine

In Game Theory, the formal modeling of gaming accounts for the interaction between the players' actions by obeying the game rules. Match gaming is algorithmically implemented according to the interaction of the strategies E^i each player applies in attempting to achieve maximum gain. We give a formal definition of strategy that will be used later to define basic tactics and some basic strategies.

Let $\hat{e} = a_1 a_2 \ldots a_n$ be a strategy, so $\hat{e}$ is a sequence of planned actions.

- $E^i = \{e_1, e_2 \ldots e_n\}$ the set of strategies.
- $E = E^{p_1} \times E^{p_2}$ is the strategy space.
- $t^i = \{t_1, t_2 \ldots t_o\} \subseteq E^i$ the set of tactics.

A FSM for Go strategies formal handling is defined as follows (see Fig. 4):

$$F_E = (\hat{A}, \hat{S}, s, \varphi, H) \quad (1)$$

$\hat{A} \subseteq A$ is a set of symbols that denote basic actions, $\hat{S} = \{s, s_0 \ldots s_l, h\}$ is the set of states, φ: $\hat{S} \times \hat{A} \rightarrow \hat{S}$, is the transition function. $H = \{h\}$ is the set of final states, $s \in \hat{S}$ is the initial state.

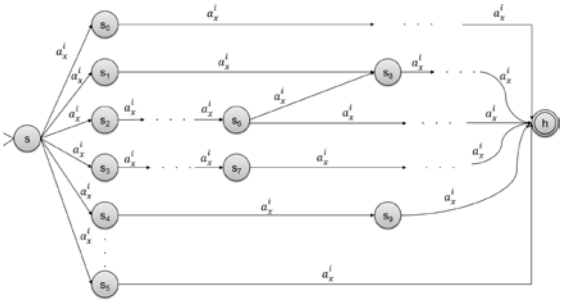

Fig. 4 FSM for handling the player' strategies from basic actions $a^i x$

This FSM computes any strategy passing by the middle state until the halt one h; the FSM is deterministic and thus, given a state and an action, there exists only one transition $a_x^{\,i}$ to the next state and each action $a_x^{\,i}$ is different in the strategy $e_x^{\,i} = a_1^{\,i} a_2^{\,i} \ldots a_p^{\,i}$. A tactic $\vec{t} \in t^i$ is defined by $\vec{t} = (a_x)$, see Fig. 5.

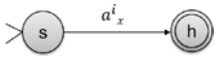

Fig. 5 FSM modeling the player's tactic a^i

See Fig. 6 for the FSM for modeling eyes. Stones holding eye conditions and halting at h_0, h_1 and h_2 are edge, lateral, and central eyes, respectively. Here, $\hat{S} = \{s, s_0, s_1, s_2, h_0, h_1, h_2\}$d $H = \{h_0, h_1, h_2\}$.

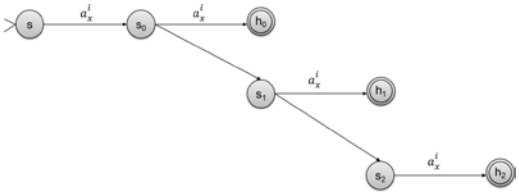

Fig. 6 FSM for eyes creation

Considering the FSM for ladders, let $\hat{S} = \{s, s_0 \ldots s_z, h\}$ and $H = \{h\}$ be states. Any string of symbols halting at h and forcing an adversary's stone in atari results in a ladder (see Fig 7):

Fig. 7 FSM for ladder creation

Regarding the FSM for nets, any string of symbols halting at h and surrounding the adversary's stones results in a net (see Fig 8):

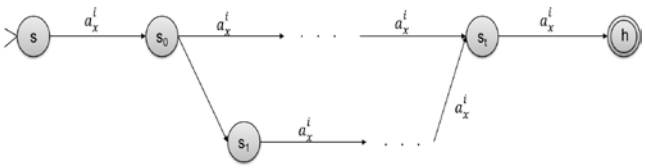

Fig. 8 FSM for net creation

From the current state of a Go match, the human player's next move should result from recognizing the Go tactics and strategies deployed by the adversary thus far in a global assessment of the match. Locally, the next node in the each player's match-game decision tree should result after an empirical visual analysis of the Go match pattern at this moment.

III. Artificial Neural Network and Monte Carlo Tree Search

The ability of NNs to find hidden relationships among the input-output mapping of pattern data makes them sufficiently powerful to deal with huge amount of combinations of forms, such as the ones emerging in Go gaming automation. Complementarily, the MCTS working on convenient search space is truly efficient for the automation of a match end step.

A. Neural Networks

The classic back-propagation NN for training on pattern recognition uses supervised learning to adjust the connection weights and enable the recognition of complex patterns. The topology of a multilayer NN is shown in Fig. 9. We use NNs for Go tactics pattern recognition as the basis for deciding the next Go actions, based on the given patterns of atari and capture conditions and on the analysis of the current state of a match.

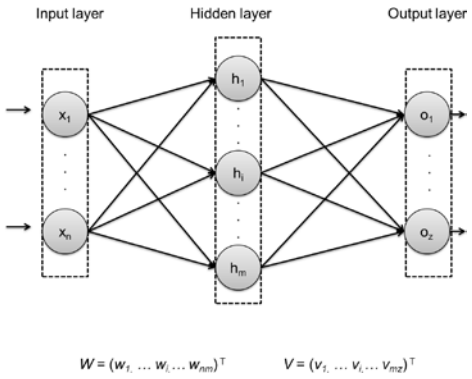

Fig. 9 example of multi-layer NN topology

For Go gaming on small boards, symbiotic adaptive neuro-evolution uses neural networks combined with evolutionary algorithms to determine the good/bad moves for a win/loss [21]. The automated Go player *Honte* [25] uses NNs in conjunction with supervised temporal differences (TD) for learning local shapes, and additional NNs for performing estimates of the value of the territorial potential. In [26], an automated Go player based on a back-propagation NN presents an architecture for learning a model from training data.

TD learning using simulation-based searches has been applied to reinforcement learning for Go gaming automation in two phases: learning and planning [15]. During the learning step, the player improves the defined policy as follows. Each

node (state) value is updated from both the MCTS simulations and the value function approximation and bootstrapping from the current node to the match end; the mean outcome of simulated episodes from real Go matches is used to value each node in a search tree and between related nodes. A TD search has been applied on Go matches over 9 × 9 boards by using naive binary features matching simple patterns of stones. Complementarily, during the planning step, the policy is improved by performing iterative simulations that start at each node [27]. A major drawback, however, is that MCTS for Go gaming automation needs too many simulations per move to obtain good results. Therefore, to overcome the lack of efficiency, the TD search integrates Go *a-priori-knowledge* to decide the next moves.

A Go machine learning approach has been developed that focuses on an evaluation function regarding scalability, from the library of local tactical patterns, the integration of patterns across the board, and the size of the board itself [28]. The automated learning is on local patterns from a library of games, by means of a recursive probabilistic Bayesian NN, the outputs of which represent local territory ownership probabilities. A combination of NNs, particle swarm optimization, and evolutionary algorithms has been used to train a board evaluator from zero knowledge [29]; the hybrid algorithm provides an evaluation of the game board through self-play. The authors claim that after experiments against the benchmark game of Capture Go, the hybrid algorithm includes and overcomes the power from the parts.

B. Monte Carlo Tree Search

Monte Carlo methods are rooted in statistical physics for the modeling of stochastic phenomena [17]. In Go gaming automation, MCTS is a best-first search technique, using stochastic simulations to estimate the value of the moves (see Fig. 10) and thus, adjust the policy towards a best-first strategy [17]. MCTS simulation values the nodes in a search tree that is the partial game tree being progressively built. The building of a tree is performed by following the *selection*, *expansion*, *simulation*, and *back-propagation* mechanisms [19].

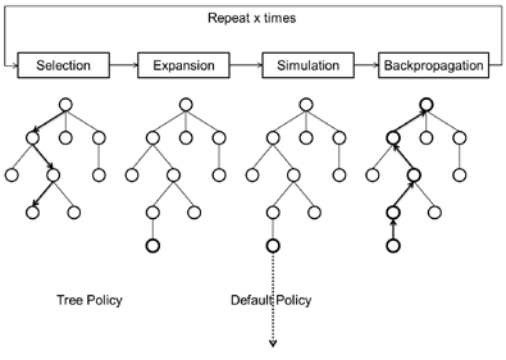

Fig. 10 MCTS process can be divided into selection, expansion, simulation, and back-propagation

The well-known Olga and Oleg Go player automation is a pioneer on the application of the Monte Carlo approach [30]. To evaluate a move from a current state *s*, many self-play simulations are performed and the value of *a* is the average value from the outcome of the simulations using a uniform random policy. Progressive pruning and the all-moves-as-first heuristic allows more rapid gaming without decreasing the Go player level. The Rapid Action Value Estimation (RAVE) extension from the MCTS algorithm [16] shares the value of actions across each sub-tree of the tree search, and the heuristic MCTS uses a function to initialize the value of the new positions in the tree search.

IV. Tactics and Strategies

In order to create a robust and reliable network, sometimes, random noise is added to training data [9]. *Don´t care* symbols are replaced by 2s, 1s or 0s in each training stage in order to preserve conditions to be a true training set for eyes, ladders, and nets, respectively. The NN error is obtained from the difference between the output from the training data and the target during iterative steps. The error is fed back repeatedly to the previous layers to modify the connections weight of nodes until a predefined tolerance or number of epochs is achieved. The NN activation of the nodes is by the sigmoid function. The number of neurons in the hidden layer is obtained experimentally and the output layer indicates the recognized patterns.

A. Tactics Pattern Recognition

For the process of pattern recognition analysis, the Go board is segmented into a *window view size* of 3 × 3 in order to identify eyes patterns. A *window view size* of 5 × 5 is used for identifying ladders and nets patterns, and as a result of the neighboring combination of these windows, bigger ladders and nets can be recognized. The NN layer of the input receives a set of board occupied points; 9 for eyes and 25 for a ladder or a net. During the training stage, the training patterns include *don´t care* symbols to represent those points that can be replaced regardless of their value. It is valuable to include *don't care* patterns in Go tactics recognition because of the non-deterministic nature of Go gaming, see example in [31].

To start the process of Go tactics pattern recognition, the positions from the 3 × 3 and 5 × 5 *windows view size* are encoded into a vector that feeds the network. When using pattern recognition to identify Go tactics in a match, the main difficulty concerns verifying that a shape really corresponds to an eye, ladder or net. The NN should check the conditions to authenticate whether the detected shape is a true Go tactic.

For eye pattern recognition, the NN tries to find similarities with the given input to any of the shapes: edge, lateral or normal eyes. If high similarities exist then the conditions of eye must be checked, i.e., there must be an empty point of space surrounded by friendly stones such that no adversary's stone may be set upon it. These conditions are verified outside the NN using verifier conditions. As in the case of eye patterns, the same procedure is applied for ladder and net pattern recognition, but with the proper ladder and net conditions.

B. Building of Strategies from Pattern Recognition

Once the Go tactics patterns such as eyes, ladders or nets, are recognized, as well as the Go gaming strategies of invasion, reduction, connection or capture, offensive/defensive strategies can be employed (see Fig. 11). Hence, based on the tactics pattern recognition, deployment heuristics for suitable defensive/offensive Go *a-priori-knowledge* strategies are available to be applied during the initial and middle steps of an automated Go match. Strategies of reduction and invasion as well as defensive strategies, address saving stones in atari or are devoted to augment the liberties of ally stones. Strategies can be constructed following the next statements, as illustrated in Fig. 12.

Defensive strategies:

- Save a stone in atari by close placement of an ally stone that eventually connects and saves it, or increasing *liberties*.
- For preventing a stone falling within risk of be captured by the adversary's next moves.

Offensive strategies:

- Interrupt the formation of adversary's eye by placing a new stone.
- Reduce *liberties* to adversary's stones, eventually placing it in atari.
- Play close to own stones, sets of stone(s) or close to stone(s) with two or more eyes to ensure high possibilities of making *connections* and *spreading* of stones.
- Capture adversary's stones by placing adversary's stones in or close to atari.

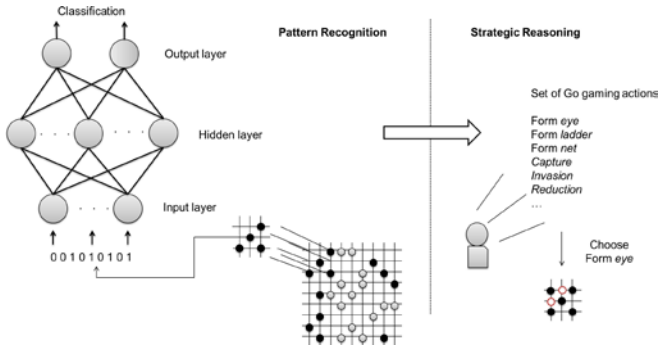

Fig. 11 example pattern recognition of a possible eye

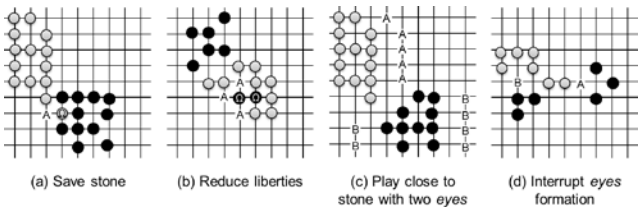

Fig. 12 Go gaming strategies: (a) Ω is in atari, but playing in point A makes a connection for saving Ω; (b) playing in points A reduces Ω liberties; (c) white/black playing to points A/B increases dominance area; (d) white/black playing to points A/B interrupts the formation of adversary's eye

V. Timely Pattern Recognition

A. Analysis of Neural Network and Monte Carlos Tree Search by stage

The huge amount of forms that occur in a Go gaming match makes the tactics patterns recognition a difficult task. Even though the recognized patterns correspond to tactics that are significant for the proposed Go strategic analysis, approach [32] tries to determine Go patterns in game records like *edge* and *corner* patterns, but few of them represent proper Go tactics patterns.

Fig. 13 shows that the highest number of recognized patterns occurs in the match middle stage. In the early and middle stage, based on the Go gaming *a-priori-knowledge* of these states, the pattern recognition and strategic reasoning work, thus a better strategic analysis of offensive/defensive Go actions is available. But when we lack of information or the board free positions arises too restrictive, that correspond to typical circumstance in late stage, the usage of MCTS on the set of free positions do choice the best to play in. Numbers in *x* axis represent Go match turns for both players.

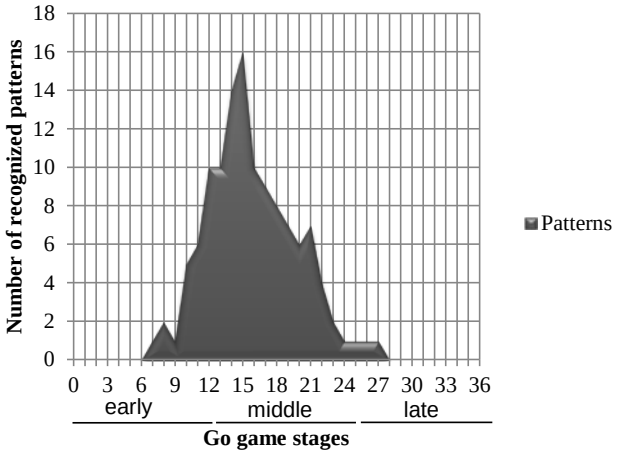

Fig. 13 number of patterns recognized through stages in a Go match

The Fig. 14 shows the elapsed time per move per player throughout the stages of a Go match. Time required per move using pattern recognition is too low and almost constant during the first and middle stage, but strongly increases in the late stage since the difficulty to recognize any pattern on the board in this stage. On the opposite, using MCTS, time spent for doing a move is too high in the early stage, comes down in the middle stage and is truly short in the late stage. Reason is that the size of search space the algorithm works at the early stage is huge, so applying MCTS is expensive and waste a lot of time; in the late stage of a Go match the search space size is small so quick to apply MCTS.

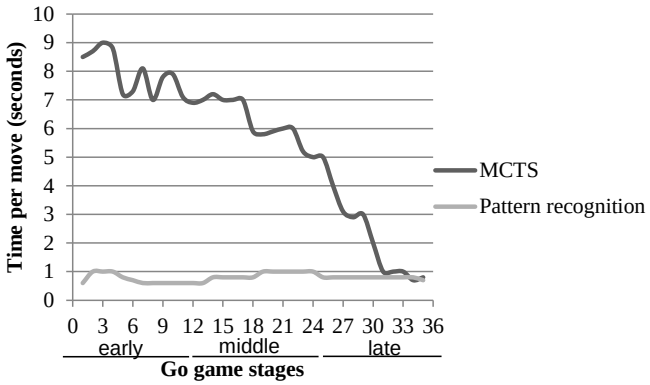

Fig.14 elapsed time per move throughout stages of a Go match

The Fig. 15 shows the percentage observed of random moves in 100 simulations. In the late stage MCTS movements are suitable since each position is quick to evaluate, so the best one up to the method is selected. In the early and middle stages MCTS is time spending, but not random pattern recognition supporting *a-priori*-known strategic Go movements are ease to apply.

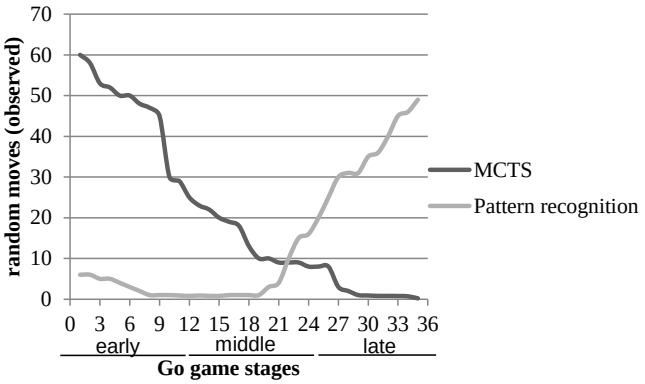

Fig. 15 elapsed time per move throughout stages of a Go match

During the early and middle stages of Go match automation, the use of pattern recognition to identify the adversary's sequence of movements is pretty efficient. By pattern recognition, the identification of the tactics and strategies followed by the adversary over these automated match stages can be achieved efficiently. From the players' experience to decide the next move, using a pattern recognition expression of *a-priori-knowledge* for local movement decision making is available. Moreover, pattern recognition serves as the basis to deploy a match gaming strategic global answer. Next, the pattern recognition for tactics and strategies is deployed.

During the latter stages of a Go match, an MCTS-based move becomes a suitable option. This is because the size of the search space has become small and the automated pattern recognition is too difficult to perform over the complex patterns on the board with the few free board positions. Actually, in the latter stages of a Go match, the deployment of *a-priori*-known strategies is hard because the free board points are too restrictive, and few board spaces make it difficult to deploy strategies. Under this circumstance, the gaming method is to perform an MCTS evaluation to play any of the free board positions. Hybrid approaches with Go *a-priori-knowledge*-based strategies are easy to deploy in the initial and middle stages of the game when few board points are occupied. By the end stages, the use of MCTS is better suited to choosing a move on the empty board positions.

B. Neural Network Go players

Our gaming simulator compromises a set of automated Go players, using either random, or pattern recognition, or strategic reasoning, or MC-Rave methods. In addition, it has a graphic interface and uses Go Text Protocol for communication with other automated players in KGS [33] or CGOS [34] Go servers (see Fig. 16).

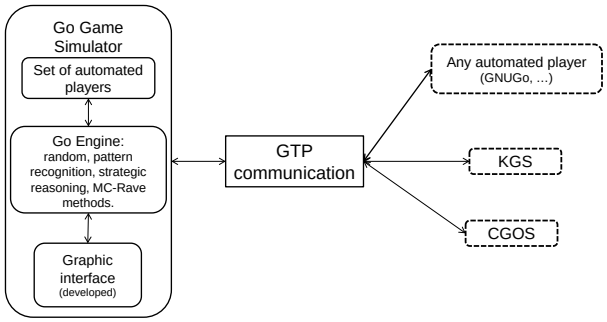

Fig. 16 Components of Go gaming simulator

In [21], the NN output positive value indicates a good move, the larger the value, the better the move. In [25], one NN previously trained from a database of Go expert players' matches, tries to imitate local Go shapes; another NN estimates safely stones and a third NN tries to find potential unoccupied territory. In [28], an NN integrates local information across the board in all directions and produces outputs that represent ownership probabilities for identifying local territory to be occupied. In [29], an NN is used as the evaluation function of the leaf nodes in a game tree, with zero expertise involved. In contrast to our approach, the use of an NN is for pattern recognition of Go tactics. For this, an NN is trained with a set of *Don't care* patterns. Once the NN is trained, the board game is segmented into *windows view size* of 3 × 3 and 5 × 5. Each *windows view* is encoded into a vector that serves as input to the NN, and the NN tries to find similarities with Go tactics, such as eyes, ladders and nets. Based on this recognition, a set of Go gaming actions is proposed. Table II summaries some aspects of the approaches based on NNs.

Table II. COMPARISON OF THE USE OF NNS FOR PLAYING GO

Approaches	Purpose	How is trained	Accuracy
N. Richard et al. 1998 [21]	To define the next move	-	-
F. Dahl 2001	Detect local	400 games	-

[25]	shape, territorial and safe group	records	
L. Wu et al. 2008 [28]	Identify local territory ownership probabilities	From a dataset played by human	-
X. Cai et al. 2010 [29]	Use to approximate the board evaluation function	Records of amateur games	-
Own approach	Tactics patterns recognition to use on offensive / defensive.	Using a set of don´t care patterns of eyes, ladders and nets forms	Above 70 %

A comparative review of the NN-based approaches for automated Go players is shown in Table III. The use of NNs for pattern recognition is essential in our proposal.

Table III. ANALYTICAL COMPARISON AMONG DIFFERENT APPROACHES

Go game proposal / Features	**Our approach**	**N. Richard et al. 1998** [21]	**F. Dahl 2001** [25]	**L. Wu et al. 2008** [28]	**X. Cai et al. 2010** [29]
Plays in small board size	✓	✓	✓	✓	✓
Plays in medium board size	✓	×	✓	✓	✓
Plays in large board size	✓	×	✓	✓	-
Board Segmentation	✓3 × 3; 5 × 5 Size	×	x	✓3 × 3 Size	✓
Multi-Players including	✓4	✓2	✓	-	-
FSM and formal languages Modeling	✓	×	×	×	×
Diversity strategies and tactics including	✓	✓	✓	✓	✓
Visual tactics recognition	✓	×	✓Detects good or bad shapes	✓	×
Visual strategies recognition	✓	×	×	×	×

Efficiency [35] is the correct way to use the available resources for doing a task, measured by runtime. In our case, this is the time taken for training the net and simulating an entire match game; a reduced runtime implies more efficiency.

Efficacy [35] is the ability to achieve the desired goals or the realization of the activities to reach the goals. In our case, this is measured by the number of frequent similar patterns obtained. See Table IV for a comparison of efficiency and efficacy of the various NN Go automation approaches.

Table IV. EFFICACY AND EFFICIENCY REPORTED ON THE DIFFERENT APPROACHES

Issue / Feature		**Our approach**	**N. Richard et al. 1998** [21]	**F. Dahl 2001** [25]	**L. Wu et al. 2008** [28]	**X. Cai et al. 2010** [29]
Efficiency	NN	19 × 19 board size, each net training spends ≈ 10 seconds	9 × 9 board size, training spends 5 days, and "the training times increase with BS quite rapidly"	-	For training a 9 × 9 board size is $O(N^4)$ and for future 19 × 19 could take months	-
Efficacy	NN	Net accuracy is above 70%.	-	-	"NN can learn territory predictions fairly well"	-

SANE [21] plays Go on a small board, cannot perform pattern recognition and needs 100 to 1000 simulations to achieve a 75% win rate over an adversary. Other approaches that use NNs, but do not present statistical results of accuracy are given in [25, 28, 35].

The results show that the strategic reasoning based on pattern recognition during the early and middle stages is appropriate because it allows the deployment of the strategically suitable moves through the deployment of previously known effective Go tactics and strategies. However, as the performance of this method reduces, MCTS is able to replace it and select moves based on the remaining free

positions on the board. In the latter stages of a Go match, the switch to MCTS becomes an efficient option, because the size of the search space has become small and automated pattern recognition is too difficult to perform with the complex patterns on the board and the few remaining free board positions. Actually, in the late Go match stage, the deployment of *a-priori*-known strategies is difficult because the free board positions are too restrictive; few board spaces make it difficult to deploy strategies. Under this circumstance, the gaming method is by performing an MCTS evaluation to play any of the remaining free board positions. Therefore, in the early and middle match stages when few board points are occupied the Go *a-priori-knowledge*-based strategies are easy to deploy. At the end of a Go match, the use of MCTS is better for determining a move based on the empty board positions.

VI. Discussion

Analysis on Go gaming automation from the complex network approach, like the one of the World Wide Web, focuses on the non-trivial topology of the network that results from a Go match [36]; the construction of a directed network given a suitable definition of related tactical Go gaming moves. By mining database matches of master level games, this approach discovered the similar patterns arising during the early stages of a Go match. In [37], the proposal for two dynamic randomization techniques is given: one for the parameters and the other for a hierarchical move generator.

The similarity between fractal formation and Go gaming patterns, is the diversity of the complex forms involved. However, a fractal formation follows a predetermined and regular pattern, but no previous regular pattern is followed by playing Go. Actually, the eyes, ladder and nets patterns are all obtained by ongoing strategies pertinent to each Go match.

Complex pattern recognition is present in Bioinformatics, which is devoted to computer and information analysis and the management of data on biological processes [38-40], particularly in determining or classifying molecular or tissue patterns as equivalent or related to some extent. Pattern recognition for Go gaming and Bioinformatics processes may advance in parallel from now on.

In computer complexity theory [41], the problems pertain to specific complexity classes by regarding certain characteristics: one major is *time*, which refers to the number of execution steps that an algorithm used to solve a problem; the other main complexity character is *space*, which refers to the amount of memory used to deal with a problem. Some complexity classes are **P**, **NP**, **PSPACE**, **and EXPTIME**. As a result of some complexity analyses of Go gaming, *experts* of the area claim that Go gaming belongs to **EXPTIME**-complete game [42], because it is an *unbounded two-player game. Unbounded games* are those in which there is no restriction on the number of moves that can be made. However, Go seems to be a bounded game because in each move a stone is placed, but there exist *capturing moves* that reopen spaces on the board. Papadimitriou [43], in one of his analyses, concluded that Go is a **PSPACE**-complete game.

Actually, being aware of what the adversary is doing helps to formulate defensive actions that inhibit her offensive actions. The inspired thinking that humans are capable of, as a result of observing the decisions other people make, applies in Go gaming through performing pattern recognition to decide on the next offensive/defensive move. The proposed NNs recognize forms that are Go tactics patterns and therefore, give relevant information to strategic decision making during the early and middle stages of a Go match.

The algorithmic analysis of Go game is paradigmatic in computer science nowadays. Moreover, the use of Artificial Neural Networks and heuristics Monte Carlo Tree Search for implementing Go tactics and strategies is a relevant contribution from computer science to some engineering and social sciences. The *control of territory* as the central problem in Go automation is direct or implicit present in a lot of problems in real life. As an instance, control on epidemic diseases may be efficiently treated using tactics of Go game.

VII. Conclusion

Formal modeling of Go tactics and strategies is by using Finite State Machines. Pattern recognition based on NN is effective during the early and middle steps of a Go match; the expert's *a-priori-knowledge* for recognizing Go eyes, ladders and nets is efficiently translated by means of NN for Go gaming automation. Actually, pattern recognition techniques are offered to build up and apply tactics and strategies during the first and middle steps of a Go match, either in defensive or offensive movements. Complementary, the huge difficulty to perform *a-priori-knowledge-based* strategies during the latter stage of a Go match, is convenient surpass by means of the use of MCTS. Go gaming, formation of diverse complex networks, as well as growing of fractals, may suggest there are underneath common problem, and solutions, in all of these topics.

Acknowledgment

Thanks are extended to the Mexican National Council of Science and Technology (CONACyT) in relation to Arturo Yee's PhD degree grant, CVU 261089.

References

[1] J. McCarthy, "AI as sport," Science, vol. 276, no. 5318, pp. 1518–1519, 1997.

[2] K. Chen, and Z. Chen, "Static analysis of life and death in the game of Go," Inf. Sci. Inf. Comput. Sci., vol. 121, no. 1–2, pp. 113–134, 1999.

[3] D. B. Benson, "Life in the game of Go," Inform. Sciences, vol. 10, no. 2, pp. 17–29, 1976.

[4] L. V. Allis, *Searching for Solutions in Games and Artificial Intelligence*. University of Limburg: The Netherlands, 1994.

[5] S. Hubalovsky, "Modeling and Computer Simulation of Real Process – Solution of Mastermind Board Game," Int. J. Math. Comput. Simulat., vol. 6, no. 1, pp. 107-118, 2012.

[6] M.P, Karpowicz, "Nash equilibrium design and price-based coordination in hierarchical systems," Int. J. Appl. Math. Comput. Sci., vol. 22, no.4, pp. 951-969, 2012.

[7] A.N. Al-Rabadi, " Qudits representations and computations of n-player many-valued quantum game, " Appl. Math. Comput., vol. 175, no. 1, pp 691-714, 2006.
[8] E.N. Chukwu, "Cooperation and competition in modeling the dynamics of gross-domestic products of nations," Appl. Math. Comp., vol. 163, no. 2, pp. 991-1021, 2005.
[9] N. Yoshiaki, *Strategic Concepts of Go*. Japan: Ishi Press, 1972.
[10] J. Kim, *Learn to Play Go*. New York: Good Move Press, 1994.
[11] E. R. Berlekamp, and D. Wolfe, *Mathematical Go: Chilling Gets the Last Point*. Massachusetts: A K Peters Ltd, 1997.
[12] P. Drake, and Y.-P. Chen, "Coevolving partial strategies for the game of go," in *Proc the International Conference on Genetic and Evolutionary Methods*, 2008, pp. 312–318.
[13] M. Müller, "Computer Go," Artif. Intell., vol. 134, no. 1–2, pp. 145–179, 2002.
[14] S. Gelly, et al., *The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions.* Communication of the ACM 55, vol. 3, pp. 106–113, 2012.
[15] D. Silver, R.S. Sutton, and M. Müller, "Temporal-difference search in computer Go," Mach. Learn., vol. 87, no. 2, pp. 183–219, 2012.
[16] S. Gelly, and D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer Go," Artif. Intell., vol. 175, no. 11, pp. 1856–1875, 2011.
[17] C. Browne, et al., "A Survey of Monte Carlo Tree Search Methods," IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no.1, pp. 1–43, 2012.
[18] J. B. Hoock, et al., "Intelligent Agents for the Game of Go," IEEE, Computational Intelligence Magazine, vol. 5, no.4, pp. 28–42, 2010.
[19] G. Chaslot, et al., "Monte-Carlo Tree Search: A New Framework for Game AI," in *Proc. Artif. Intell. Interact. Digital Entert*, 2008.
[20] A. Yee, and M. Alvarado, "Well-Time pattern recognition in Go gaming automation, " in Proc. Mathematical Methods and Computational Techniques in Science and Engineering (MMCTSE). 2014.
[21] N. Richards, D.E. Moriarty, and R. Miikkulainen, "Evolving Neural Networks to Play Go," Appl. Intell., vol. 8, no.1, pp. 85–96, 1998.
[22] E. D. Werf, H.J. Herik, and J.H.M. Uiterwijk, "Learning to Estimate Potential Territory in the Game of Go," in *Proc. Computers and Games*, H.J. Herik, Y. Björnsson, and N. Netanyahu, Editors., Springer Berlin Heidelberg: Ramat-Gan, Israel, 2006, pp. 81–96.
[23] B. Bouzy, and T. Cazenave, "Computer Go: an AI Oriented Survey," Artif. Intell., vol. 132, no.1, pp. 39–103, 2001.
[24] E.H.J. Nijhuis, *Learning Patterns in the Game of Go*, in *Artificial Intelligence*, Universiteit van Amsterdam: Amsterdam, 2006.
[25] F. A. Dahl, *Honte, a go-playing program using neural nets*. Machines that learn to play games, Nova Science Publishers, 2001 pp. 205–223.
[26] L. Jianming, Z. Difei, and L. Rui, "Improve Go AI based on BP-Neural Network," in *Cybernetics and Intelligent Systems, IEEE Conference on*, 2008.
[27] C. Fellows, Y. Malitsky, and G. Wojtaszczyk. "Exploring GnuGo's evaluation function with a SVM," in *Proc. of the 21st national conference on Artificial intelligence*, Boston, Massachusetts: AAAI Press, 2006.
[28] L. Wu, and P. Baldi, "Learning to play Go using recursive neural networks," Neural Networks, vol. 21, no.9, pp. 1392–1400, 2008.
[29] X. Cai, G.K. Venayagamoorthy, and D.C.W. II, "Evolutionary swarm neural network game engine for Capture Go", Neural Networks, vol. 23, no. 2, pp. 295–305, 2010.
[30] B. Bouzy, and B. Helmstetter, *Monte-Carlo Go Developments*. in *Advances in Computer Games*, Springer US, 2004, pp. 159–174.
[31] A. Yee and M. Alvarado,"Pattern Recognition and Monte-CarloTree Search for Go Gaming Better Automation," in Proc. Advances in Artificial Intelligence -IBERAMIA, Springer Berlin Heidelberg, 2012, pp. 11-20.
[32] Y. Shi-Jim, et al. "Pattern Matching in Go Game Records," presented at Second International Conference on Innovative Computing, Information and Control, 2007.
[33] *KGS Go Server*. Available from: http://www.gokgs.com/.
[34] *CGOS*. Available from: http://cgos.boardspace.net/.
[35] F.P. Miller, A.F. Vandome, and J. McBrewster, "Computational Complexity Theory," Saarbrücken, Germany: VDM Publishing House Ltd. 2009.
[36] B. Georgeot, and O. Giraud, "The game of go as a complex network," Europhysics Letters, vol. 97, no. 6, 2012.
[37] K.-H. Chen, "Dynamic randomization and domain knowledge in Monte-Carlo Tree Search for Go knowledge-based systems," Knowledge-Based Systems, vol. 34, pp. 21–25, 2012.
[38] M. Hue, et al., "Large-scale prediction of protein-protein interactions from structures," BMC Bioinformatics, vol. 11, no. 1, pp. 1–9, 2010.
[39] M. D. Ritchie, et al., "Genetic programming neural networks: A powerful bioinformatics tool for human genetics," Appl. Soft Comput., vol. 7, no. 1, pp. 471–479, 2007.
[40] O. Wolkenhauer, et al., "SysBioMed report: Advancing systems biology for medical applications," Systems Biology, IET, vol. 3, no. 3, pp. 131–136, 2009.
[41] A. Sanjeev, and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge: Cambridge University Press, 2009.
[42] R. A. Hearn, *Games, Puzzles, and Computation*, in *Department of Electrical Engineering and Computer Science*. Massachusetts Institute of Technology: Cambridge, 2006.
[43] C. H. Papadimitriou, *Computational Complexity*. Reading, Massachusetts Addison Wesley, 1993.

Arturo Yee Rendón received his M. Sc. degree in Computer Science from the Computer Science Department in the Center of Research and Advanced Studies (CINVESTAV-IPN), Mexico, where nowadays is researching on his Ph. D. thesis focus on strategic reasoning for games playing. He received his bachelor degree in Computer Science from the Autonomous University of Sinaloa, Mexico, being distinguished with the Best Student Award in his career promotion.

Matías Alvarado is a research scientist in the Computer Science Department of the Center of Research and Advanced Studies (CINVESTAV-IPN), México. He is a member of the Mexican Academy of Sciences. He is doctor in Science Mathematics from the Technical University of Catalonia. His current major research interest is the mathematical modeling of strategic reasoning and the computer simulation of strategic decision making. He has published around 70 papers, most of them on Decision Making methods, the last ones focusing in selection of strategies in sports and board games.