LS-Draughts: Using Databases to Treat Endgame Loops in a Hybrid Evolutionary Learning System

Henrique Castro Neto, Rita Maria Silva Julia and Gutierrez Soares Caixeta
Computer Science Department, Federal University of Uberlândia
Brazil

1. Introduction

The Reinforcement Learning methods have been a subject of great interest in the machine learning area, since it does not require an intelligent "professor" to provide training examples. That is why it is a suitable tool for dealing with complex domains where it is hard or even impossible to obtain such examples (Russell & Norvig, 2003). Among the Reinforcement Learning methods, one can be stood out: the TD learning methods. They are widely used with highly efficient results, including in the construction of agents capable of learning to play Draughts, Chess, Backgammon, Othello, GO or other games (Samuel, 1959; Lynch, 1997; Lynch & Griffith, 1997; Neto & Julia, 2007; Schaeffer et al., 2001; Thrun, 1995; Tesauro, 1994; Leuski, 1995; Schraudolph et al., 2001 and Epstein, 2001). Such agents have demonstrated that the games are, undoubtedly, a very suitable domain to study and to check the efficiency of the main techniques of machine learning.
As an example of good automatic player agents, the LS-Draughts - (Neto & Julia, 2007) can be cited. It is a Draughts learning system based on Mark Lynch's NeuroDraughts player (Lynch & Griffith, 1997) which uses three important tools in machine leaning: Genetic Algorithms (AGs), Artificial Neural Network (ANN) and Temporal Differences (TD) Reinforcement Learning methods. It also adopts the NET-FEATUREMAP mapping technique to represent the game board states. This mapping represents the game board states by means of a set of functions – called features – that capture relevant knowledge about the domain of draughts and use this knowledge to map the game board in the input of an Artificial Neural Network. Using Genetic Algorithms, the LS-Draughts extends NeuroDraughts and automatically generates a concise and efficient set of features which is relevant to represent the game board states and to optimize the training of the draughts player agent (NeuroDraughts uses a fixed and manually defined set of features). The core of LS-Draughts consists of an Artificial Neural Network whose weights are updated by the Temporal Differences Reinforcement Learning methods. The Network output corresponds to a real number (prediction) that indicates to what extent the board sate represented by features in the Network input is favorable to the agent. The agent is trained by self-play coupled with a cloning technique. The minimax algorithm is used to choose the best action to be executed considering the current game board state. LS-Draughts shows that the GAs can be an important tool for improving the general performance of automatic players.

Source: Theory and Novel Applications of Machine Learning, Book edited by: Meng Joo Er and Yi Zhou,
ISBN 978-3-902613-55-4, pp. 376, February 2009, I-Tech, Vienna, Austria

Furthermore, GAs presented a good performance along with another important machine learning technique: the Temporal Differences Reinforcement Learning methods. Despite of the fact that LS-Draughts improved the general performance of NeuroDraughts, it did not manage to lessen its endgame loop problems (Neto & Julia, 2007).
In order to attack this problem, here the authors present an extended version of LS-Draughts where the endgame databases of the exceptional automatic draughts player Chinook (Schaeffer et al., 1996; Schaeffer, 1997) have been added to.
The main purpose of including the endgame databases of Chinook into the extended LS-Draughts is to try to answer two important questions:

1. Will the addition of the endgame databases into the original LS-Draughts contribute, in fact, for improving its general performance?
2. Will the use of the endgame databases help to decrease the rate of endgame loops in the original LS-Draughts? (This problem occurs in NeuroDraughts as well).

Finally, a tournament was executed between the available player of NeuroDraughts, the best player of the original LS-Draughts and the best player generated by the extended version of LS-Draughts proposed here. Furthermore, the rates of endgame loops occurred during the training process of these three players were also estimated. The results obtained show that the insertion of endgame databases into LS-Draughts produced a much better player and decreased significantly the rate of endgame loops.

2. Temporal differences methods in games

This section explains how TD Reinforcement Learning methods can be used along with minimax search by a player Neural Network. First, the Network is rewarded for a good performance (that is, it receives from the environment a positive reinforcement corresponding to the endgame state, in case of victory) and it is punished for a bad performance (it receives from the environment a negative reinforcement corresponding to the endgame, in case of defeat). For all the intermediate game board states (between the starting board and the final board) represented in the input layer of the Network, as no specific reward is available, the TD mechanism calculates the prediction P of victory by means of the following equation:

$$P = g(in^{output}), \tag{1}$$

where g is the hyperbolic tangent function and in^{output} is the local induced field on the neuron of the Network output layer (Haykin, 1998; Lynch, 1997). It means that the value of P depends on the Network weights. Then, a prediction P corresponds to a real number belonging to the interval [-1,1] that indicates how much the game board state represented in the Network input is favorable to the agent. Each time the agent must move a piece, the minimax algorithm is used to build a depth *n* breadth-first search tree whose root S is the current state (resultant from the last opponent move), whose depth 1 nodes correspond to the states resultant from all possible moves available to the agent considering the state S, and so on for the depth 2, depth 3, …, depth *n*, which correspond to all possible states resulting from the opponent's and agent's later moves up to the depth *n* level. The Network calculates, then, the predictions P for each depth *n* sate. Finally, these values will be returned to the minimax algorithm in order to allow it to indicate to the agent which is the best action to choose and to execute in S. Whenever the agent executes a move, the Network weights are updated according to equation 2 (Sutton, 1988):

$$\Delta w_t = \alpha (P_t - P_{t-1}) \sum_{k=1}^{t-1} \lambda^{(t-1)-k} \nabla_w P_k , \tag{2}$$

where P_t is the prediction corresponding to the current game board state, P_{t-1} is the prediction corresponding to the previous game board sate, each P_k represents the prediction corresponding to an earlier game board state, α is the learning rate (defined according to how fast the system will update the Network weights), λ is a constant defined according to how much the system will consider the impact of an earlier state P_k in the weight updating process and $\nabla_w P_k$ corresponds to the partial derivate of P_k with respect to the variable w (weight).

A small revolution in the field of Reinforcement Learning occurred when Gerald Tesauro presented his training results obtained by applying the TD methods to an evaluation function (Tesauro, 1992; Tesauro, 1995). Tesauro's program, TD-Gammon, is a backgammon player that, in spite of having very little knowledge about backgammon, is able to play as efficiently as the greatest world players (Tesauro, 1994). The principles of TD methods were first applied by Samuels, who pioneered the idea of updating evaluations based on successive predictions in a checker program (Samuel, 1959). Another very successful work with the TD methods is that proposed by Jonathan Schaeffer and other researchers (Schaeffer et al., 2001) that produced a detailed comparative study between an evaluation function trained manually by experts (which is the case of the current draught champion CHINOOK (Schaeffer et al., 1996)) and an evaluation function trained by the Temporal Differences methods. This analysis showed that the self-learning strategy along with the TD methods is an efficient tool to produce automatic agents able to play with a high level of performance.

Other works that obtained good results with the TD method are: Mark Lynch (Lynch, 1997), Neto and Julia (Neto & Julia, 2007), Thrun (Thrun, 1995), Leuski (Leuski, 1995), Schraudolph (Schraudolph et al., 2001), Baxter (Baxter et al., 1998) and Levinson and Weber (Levinson & Weber, 2002).

3. Evolutionary computation in games

Evolutionary Computation is an area of Computer Science which uses ideas from biological evolution to solve computational problems. Many such problems require searching through a huge space of possibilities for solutions, such as the classification task in data mining, the selection of a collection of rules (or actions) that will control a robot as it navigates in its environment, or the job-shop scheduling task. Such computational problems often require a system to be adaptive – that is, to continue to perform well in a changing environment.

The basis for Evolutionary Computation is the schema theory modeled mathematically by Holland (Holland, 1992). The schema theory is inspired on the principle of survival of the fittest individuals of the Darwin's theory of natural selection, where the fittest individuals are selected to produce offspring for the next generation. In the context of search, individuals are candidate solutions to a given search problem. Hence, reproduction of the fittest individuals means reproduction of the best current candidate solutions. Genetic operators such as selection, crossover and mutation generate offspring from the fittest

individuals. One of the advantages of Evolutionary Computation over "traditional" search methods is that the former performs a kind of global search using a population of individuals, rather than performing a local search. The global search methods are more vigorous than the local search methods to avoid to be trapped into a local maximum. There are several approaches that have been followed in the field of Evolutionary Computation. The general term for such approaches is evolutionary algorithms. The most widely used form of evolutionary algorithms is Genetic Algorithms, which was the main focus of the original version of LS-Draughts (Neto & Julia, 2007). Other common forms of evolutionary algorithms are Evolution Strategies, Evolutionary Programming and Genetic Programming (Mitchell & Taylor, 1999).
The application of Evolutionary Computation in games has helped to produce very good player agents, principally as an alternative paradigm to the conventional training process. David Fogel, using evolutionary algorithms to update the weights of the chess player Multilayer Neural Network corresponding to his best chess player BLONDIE25 (Fogel et al., 2004), as well as to update the weights of the draught player Multilayer Neural Network corresponding to his best draught player ANACONDA (Fogel & Chellapilla, 2002), proved the usefulness of the evolutionary algorithms as a training tool, once BLONDIE25 and ANACONDA obtained the titles of master in chess and expert in draughts, respectively, due to their excellent performance in international tournaments.
Fogel also tested his player ANACONDA against an ancient good version of CHINOOK . The former obtained a better performance: 4 victories, 3 defeats and 3 draws.

4. Temporal differences x evolutionary computation

Paul Darwen demonstrates in (Darwen, 2001) that the TD methods are more efficient than Evolutionary Computation methods to train backgammon player agents implemented as nonlinear systems, since the former methods requires only a few hundred thousand training games to produce a good player, whereas the later ones would require billions of training games to obtain the same performance. On the other hand, Darwen showed that Evolutionary Computation is more efficient than TD methods to train backgammon player agents implemented as linear systems. The same seems to be also valid for draughts domain. For instance, ANACONDA needed 126.000 training games to present the same performance obtained by Schaeffer's world champion player CHINOOK – trained manually – after only 10.000 training games.
Inspired by Darwen's results, Neto and Julia demonstrated in (Neto & Julia, 2007) that GAs can be an important tool for improving the general performance of draughts player Neural Networks trained by the Temporal Differences methods. Their best draughts player obtained a better performance in relation to the Mark Lynch's NeuroDraughts player: 2 victories and 5 draws. However, despite of the fact LS-Draughts has improved the general performance of NeuroDraughts, it presented endgame loop problems. For example, analysing the 5 draws against NeuroDraughts, in 2 of them the best player of LS-Draughts could easily have won if it was able to detect endgame loops. In order to solve this problem, as mentioned above, the authors present, in the next section, the architecture and the implementation of the extended version of LS-Draughts (including the GA of the original version) where the endgame databases of the draughts player Chinook have been added to. Next, the problem of the endgame loops found in the original LS-Draughts will be shown. Finally, this chapter shows the results obtained by the extended version of LS-Draughts.

5. The LS-Draughts with endgame databases

The extended LS-Draughts is a learning system whose main objective is to generate a draught player agent able to play draughts on a high performance level. In order to cope with this objective, the LS-Draughts extends the Mark Lynch's NeuroDraughts player (Lynch, 1997; Lynch & Griffith, 1997) by the adiction of the following modules:

1. An automatic feature generation module whose purpose is to automatically generate, by means of Genetic Algorithms, a concise set of features which are essential for representing the game board states and to optimize the training of the LS-Draugths' player agents. More concisely, each agent consists of an Artificial Neural Network whose weights are updated by the Temporal Differences methods (Neto & Julia, 2007);
2. An endgame database module whose purpose is to anticipate the result of the game (victory, defeat or draw) for draughts board states with up to eight pieces on the board. By adding this endgame database, the extended LS-Draughts tries to improve the adjustment of the Neural Network weights through perfect information retrieved from the database indicating predictions of victory, defeat or draw (instead of using heuristic information). Indeed, this procedure tends to let the evaluation function of the extended LS-Draughts more efficienty and more accuraty, reducing, therefore, the rate of endgame loops.

The new architecture of LS-Draughts is indicated in figure 1. The figure shows, through a flowchart of arrows enumerated from 1 up to 14, an overview of total training process of LS-Draughts, including the six modules. The architecture and the flowchart are explained below.

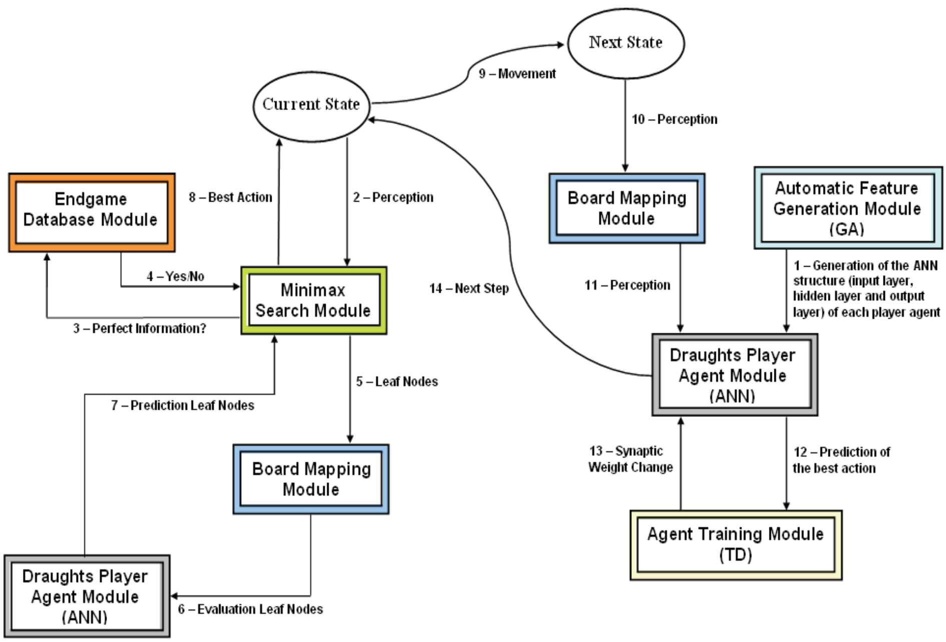


Fig. 1. LS-Draughts' new architecture, with endgame databases.

The system is composed of six main modules:

1. **The Automatic Feature Generation Module or GA:** corresponds to Genetic Algorithms that generate T_p individuals which represent subsets of every available features in the NET-FEATUREMAP mapping (the same mapping technique used by Lynch in NeuroDraughts);
2. **The Draughts Player Agent Module (ANN):** corresponds to a Multilayer Neural Network whose input layer represents a game board state. The Network output corresponds to a real number (prediction) that indicates to what extent the input state is favorable to the agent;
3. **The Board Mapping Module:** the draughts game board is implemented in an array of 32 positions, which each position represents a specific square of the draughts board. The mapping used by LS-Draughts is the NET-FEATUREMAP (more details about this kind of mapping can be found in (Lynch, 1997)). The role of this module is to represent a game board state (or array of 32 positions), in the Neural Network input, by means of a set of functions called features;
4. **The Minimax Search Module:** the role of this module is to select the best action to be executed by the agent according to the current game board state. The classical minimax algorithm can be found in (Russel & Norvig, 2003). Section 5.4.2 presents the pseudo-code corresponding to the search algorithm here proposed, which combines the minimax algorithm and the use of endgame databases;
5. **The Agent Training Module or TD:** corresponds to the learning process of the player agent or ANN. This module uses Temporal Differences Reinforcement Learning methods and self-play with cloning as training strategy;
6. **The Endgame Database Module:** the use of endgame database (Lake et al., 1994; Schaeffer et al., 1996) reduces the sequence of necessary moves, from the initial game board, in order to reach a position with defined theoretical value, that is, victory, defeat or draw. The endgame databases used by LS-Draughts are available on http://www.cs.ualberta.ca/~chinook/ and the library of functions that allows to access to these databases is available on http://pages.prodigy.net/eyg/checkers/kingsrow.htm.

As cited in (Neto & Julia, 2007), the learning process of LS-Draughts is similar to the one performed by NeuroDraughts (this latter corresponds to second, third, fourth and fifth modules of the new architecture of LS-Draughts). However, the fifth module of LS-Draughts modifies the training process of NeuroDraughts in the following way: in the former, innumerous individuals – that is, innumerous sets of features – are trained, whereas in the latter, just one is trained. Furthermore, the first and sixth modules extend the Mark Lynch's player. The first module automatically generates, by means of GAs, feature sets that tend to represent, efficiently, the game board states in order to produce good draught players. The sixth module uses the endgame databases of Chinook in order to reduce endgame loop problems and to improve the learning of the individuals. Note that this new version of LS-Draughts extends the original version only by the addiction of the sixth module. The interaction between the second, fourth and fifth modules have already been described in the section 2. More details about these can be found in (Lynch, 1997).

An overview of the flowchart of arrows indicated in the figure 1 is presented as follow:

1. **Automatic Feature Generation Module (GA) → Generation of the ANN structure (input layer, hidden layer and output layer) of each player agent → Draughts Player Agent Module (ANN):** firstly, the GA generates T_p individuals which represent subsets of every available features in the NET-FEATUREMAP mapping. Next, each individual is introduced in the input of the Artificial Neural Network corresponding to it. Consequently, T_p draughts player Neural Networks (or draughts player agents) are produced here.
   Henceforth, the next steps of the flowchart of arrows (enumerated from 2 up to 14) will refer to the training process of the extended LS-Draughts considering only one individual, that is, just one player Neural Network. Therefore, in practice, during the training process of LS-Draughts, the next steps should be repeated for each individual associated to its player Neural Network, that is, T_p times;
2. **Current State → Perception → Minimax Search Module:** each time the player agent must move a piece, the minimax algorithm is used to build a depth *n* breadth-first search tree whose root *S* is the current state. So, the parameters of the Minimax Search Module are the depth *n* and the current state *S*;
3. **Minimax Search Module → Perfect Information? → Endgame Database Module:** before that the Minimax Search Module can build the game search tree for the current state *S*, it checks, through the Endgame Database Module, whether *S* belongs to the endgame databases;
4. **Endgame Database Module → Yes/No → Minimax Search Module:** if true, the Minimax Search Module goes to the step 8. Otherwise, it carries on next step;
5. **Minimax Search Module → Leaf Nodes → Board Mapping Module:** after building the game search tree, the Minimax Search Module checks whether all of the leaf nodes of this tree belong to the endgame databases. If true, then the Minimax Search Module can compute the game theoretic value (victory, defeat or draw) using the perfect knowledge of the endgame databases, instead of using the heuristic evaluation function. So, if all leaf nodes belong to the databases, the Minimax Search Module gets their corresponding theoretic values and goes to the step 8. Otherwise, for each leaf node that does not belong to the endgame databases, the Board Mapping Module is called for mapping the game board state associated to.
   Note that the theoretic value is obtained through the Endgame Database Module and the heuristic value is obtained through the ANN (Draughts Player Agent Module). The steps 6 and 7 below show the latter situation;
6. **Board Mapping Module → Evaluation Leaf Nodes → Draughts Player Agent Module (ANN):** the Board Mapping Module maps the game board states, associated to each leaf node of the game tree into the input layer of the player Neural Network;
7. **Draughts Player Agent Module (ANN) → Prediction Leaf Nodes → Minimax Search Module:** for each leaf node, represented in the input layer of the player Neural Network, a prediction *P* of victory is calculated. This prediction *P* corresponds to a real number belonging to the interval [-1,1] that indicates how much the game board state is favorable to the agent;
8. **Minimax Search Module → Best Action → Current State:** the values that are returned to the minimax algorithm (corresponding to the theoretic values of the endgame databases and/or corresponding to the heuristic values of the prediction of the player

Neural Network) are used in order to indicate to the agent which is the best action to choose and to execute in the current state *S*;

9. **Current State → Movement → Next State:** the best action is executed and the game board current state is changed to the next state *S′*;
10. **Next State → Perception → Board Mapping Module:** the Board Mapping Module is called for mapping the game board state *S′* resultant of the best action executed by agent;
11. **Board Mapping Module → Perception → Draughts Player Agent Module (ANN):** the Board Mapping Module maps the new state *S′* in the input layer of the player Neural Network;
12. **Draughts Player Agent Module (ANN) → Prediction of the best action → Agent Training Module (TD):** the TD mechanism calculates the prediction *P′* of victory for the new state *S′* (represented in the input layer of the player Neural Network) by means of the equation 1 showed in the section 2;
13. **Agent Training Module (TD) → Synaptic Weight Change → Draughts Player Agent Module (ANN):** the new prediction *P′* calculated by TD mechanism is used for updating the synaptic weights of the player Neural Network according to equation 2 of the section 2;
14. **Draughts Player Agent Module (ANN) → Next Step → Current State:** the flowchart returns to the step 2 and repeats the whole flow until the end of the training game.

In the next subsections, the authors present the structures of the first and the sixth modules which characterize the LS-Draughts. As shown before, each individual generated by the first module will be attached to a neural network that will learn by means of training games guided by the TD methods.

5.1 Population and individual encoding in the LS-Draughts

Each individual in the population is encoded as a binary chromosome whose length is 15 genes. The binary representation indicates whether a determined feature F_i occurs or not in the gene G_i, where $i \in \{1,2,3,\ldots,15\}$, as shown in figure 2.

F1	F2	F3	F4	F5	F6	F7		F14	F15
1	1	0	0	1	1	1	...	1	0
G1	G2	G3	G4	G5	G6	G7		G14	G15

Fig. 2. Example of an individual encoding in the population.

Table 1 shows the 15 features to be represented in the genes. Each integer number in column BITS corresponding to a feature F_i indicates the quantity of neurons that will be reserved to represent F_i in the input layer of the Network.

In this chapter, the GA population is composed of 50 individuals, that is, T_p =50. Therefore, the population will be formed by 50 chromosome structures (or individuals) where each of them will be associated to a Neural Network. These 50 individuals will evolve within the GA along 50 generations (the original LS-Draughts evolved them along 30 generations).

FEATURES	BITS
F1: PieceAdvantage	4
F2: PieceDisadvantage	4
F3: PieceThreat	3
F4: PieceTake	3
F5: Advancement	3
F6: DoubleDiagonal	4
F7: Backrowbridge	1
F8: Centrecontrol	3
F9: XCentrecontrol	3
F10: TotalMobility	4
F11: Exposure	3
F12: KingCentreControl	3
F13: DiagonalMoment	3
F14: Threat	3
F15: Taken	3

Table 1. Set of features used by LS-Draughts.

The individuals will be generated according to the following steps:

1. All the 50 individuals of the first generation GE_0 are generated by a randomly binary activation (1 or 0) of their genes. Next, each of these individuals (which represents a game state) is introduced in the input of the Network that corresponds to it. The 50 Networks produced are trained (the training process will be discussed later). After this, LS-Draughts starts a tournament involving the 50 available trained Networks. At the end of the tournament, an evaluation (or fitness) is calculated for each individual based on its performance during the tournament (as detailed in the subsection 5.4). Next, the 50 individuals of GE_0 are passed as parents to the next generation (generation GE_1);
2. All the 50 individuals of the remaining 49 generations GE_i, where $1 \le i \le 49$, are generated as described below: 50 new individuals are generated by applying the genetic operators of crossover and mutation to 25 pairs of individuals chosen by a stochastic tournament selection process over the 50 parents received from the generation GE_{i-1}. Next, 50 new Networks associated to these 50 new individuals are trained. After this, LS-Draughts starts a tournament involving the 100 available trained Networks (50 corresponding to GE_{i-1} and 50 corresponding to GE_i). At the end of the tournament, an evaluation (or fitness) is calculated for each individual based on its performance during the tournament. The 50 individuals which present the best fitness will be passed as parents to the next generation GE_{i+1}, and so on.

5.2 Individual selection and application of genetic operators

The selection method used by LS-Draughts to select the parents in order to apply the genetic operators is the stochastic tournament whose tour is 3 (Mitchell & Taylor, 1999). For each 2 parents selected by the stochastic tournament selection, two new children are generated. The crossover method used is the simple crossing of genes (one-point crossover) with crossover probability = 100%. The mutation probability rate – P_{mut} – used is 0.3. Thus, in each individual, 5 genes are chosen randomly to be modified by mutation.

5.3 Neural Networks training

The multilayer Network associated with an individual I_i has N_A neurons in the input layer, where N_A represents the quantity of bits associated to the active genes (digit 1) in I_i. The hidden layer and the output layer have 20 neurons and 1 neuron, respectively. For example, in figure 3, the Network attached to the individual M will utilize only the features F_1, F_2 and F_{14} (corresponding to the 3 active genes) in order to represent the board state in the input Neural Network. Consequently, as shown in table 1, N_A is equal to 11 in the Network that represents the individual M. The initial weights of the Neural Network linked to individual I_i are generated randomly between -0.2 and +0.2 and the bias term is fixed as being 1. This process is repeated for every individual I_i, where $i \in \{1, 2, 3, ..., 50\}$.

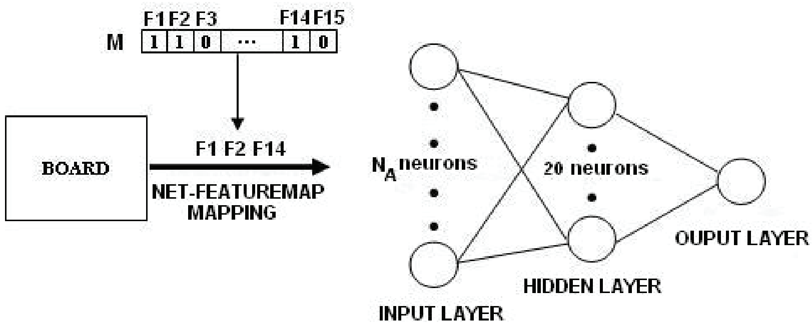


Fig. 3. Selection of active features of an individual M to define the NET-FEATUREMAP mapping which the attached neural network will use in the training.

After the generation of a Neural Network for each I_i, where $i \in \{1, 2, 3, ..., 50\}$, LS-Draughts starts the training phase of the 50 Neural Networks attached to the 50 generated individuals. In the training games, the Networks learn by reinforcement, as described in the section 2.

The training of each Neural Network consists of a group of 4 sessions of 400 training games, where the Network plays a half of these 400 games as black player (black pieces) and the other 200 games as red player (red pieces). Such a strategy has the purpose of training the agent for diversified situations, once the features establish constraints that are related to the color of the pieces (for example: feature F_9 indicates the number of red pieces in the center of the game board).

Before starting the ten training session by self-play, a copy of the Neural Network net_i attached to the individual I_i is made, thus producing the clone Network $cnet_i$. Next, net_i and $cnet_i$ play the 200 games which correspond to the first session. During these games, only the weights of net_i are updated. At the end of the first session, two test-games are played to check whether the new Network net_i corresponding to the individual I_i (remember that the original weights of net_i were updated during the games) became better than its clone. If it is true, the $cnet_i$ weights are replaced by the net_i weights and the next session begins involving the players net_i and the modified $cnet_i$ (which are, in fact, equivalents). Otherwise, $cnet_i$ is not modified and the next session begins involving the players net_i and $cnet_i$. It is interesting to point out that in these test-games the strategy of changing the color corresponding to each player is adopted, that is, each network plays one game-test as red pieces and the other one as black pieces. This process is repeated until the end of the fourth session. Note that, in all these games, both player Networks use the same strategy to choose the best action described in the section 2.

Considering the possibility that it is not reasonable to guarantee that the best network obtained in the end of the fourth session is really the best one (in case it has been specialized to beat only its last clone during the training process), a small tournament is realized between the final network obtained and all its clones generated in the four sessions. Finally, the winner of this tournament is considered as being the best network corresponding to the individual I_i that has been trained.

The input parameters utilized to train an individual I_i are: a game board file containing the initial game board settings and the minimum score required to allow the replacement of the weights of a clone by the weights of the Network from which it originated; the learning rate α_1 corresponding to the first layer ($\alpha_1 = 1/N_A$); the learning rate α_2 corresponding to the hidden layer ($\alpha_2 = 1/20$); and, finally, the λ (see section 2) value ($\lambda = 0.1$).

5.4 Using endgame databases in the neural networks training

Considering the depth-first search algorithm, the time complexity is $O(b^m)$ for a state space with branching factor *b* and maximal depth *m* (Russell & Norvig, 2003). This means that the state space grows exponentially with depth. For large state spaces, a good solution for this problem consists of combining the depth-first search with retrograde analysis techniques based on information that is eventually stored in Databases (DBs). In other words, whenever the depth-firth search tries to evaluate a node N_1, before evaluating it, the algorithm will check whether the DB stores information about N_1. If it does, this piece of information will free the search algorithm from the burden of evaluating N_1. The state space of Draughts is approximately composed of $5 * 10^{20}$ distinct board states. Even considering recent computational resources, it is impracticable to traverse such a state space by adopting the depth-first search strategy. Therefore, the use of endgame DBs in Draughts strongly improves the general performance of player agents. In fact, even if the current board state *S* that is being evaluated is distant from endgame board states, some of its descendents may already be in the DB, what may limit the search depth to the level of these descendents. The great success and efficiency of Chinook, for example, is mainly based on its endgame DBs (Lake et al., 1994). Considering the large dimension of the Draughts state space, the construction of its endgame DBs is a hard task. In fact, in Chinook, the efforts to build the DBs have begun in 1989 and, since that year, almost continuously, several computers have worked exclusively to cope with this activity (Schaeffer et al., 2007). In 1992, there were more than 200 computers simultaneously working in the construction of these DBs.

Nowadays, they store information about all game board states that comprise 10 or less pieces. Particularly, for each of these states, the DBs indicate whether it corresponds to a win, a loss or a draw state. Considering the arguments above, in order to improve its search method, the extended LS-Draughts combines the minimax algorithm with a subset of endgame DBs of Chinook, which allows it to find the best movement spending much less time. During the search process, whenever a board state *S* is found in the DB, the extended LS-Draughts, instead of using its heuristic evaluation function to calculate the prediction corresponding to *S*, retrieves from the DB its exact value. In this case, the extended LS-Draughts does not need to evaluate any of the descendant nodes of *S* in the search tree, what correspond to a relevant simplification in the minimax search process. The combination between minimax and endgame DBs produces an efficient search method which simplifies the search tree and obtains more precise results, since the predictions of win, loss or draw retrieved from the DBs are exact (Schaeffer et al., 2002). The next subsections present how to construct endgame DBs and detail how the extended LS-Draughts uses them.

5.4.1. How to build endgame DBs

The previous section showed that retrograde analysis is an efficient tool to improve the search process. It has also been successfully applied in the construction of DBs for several games (Lake et al., 1994; Schaeffer et al., 2007; Gasser, 1990; Gasser, 1996; Romein & Bal, 2002; Romein & Bal, 2003). As the construction of DBs for games requires many resources of memory, execution time and input/output (I/O), the same techniques used to implement them can be also used to solve several problems in Mathematics and other related sciences where an optimal solution must be found in large state spaces. To construct DBs corresponding to Draughts board states with *n* pieces, the state space to be analysed will be a graph that may be a cyclic one (Diestel, 2000). The board states are represented in the nodes of the graph. The DBs for game boards composed of *n* pieces are calculated by means of an iterative algorithm which uses the results obtained for the DBs of *1*, *2*, …, (*n – 1*) pieces previously calculated. Then, the DB for the board states with only 1 piece (terminal states) must be calculated first. Note that, in this case, according to the rules of Draughts, the player which owns the remaining piece is the winner. The algorithm must enumerate all the 120 possible terminal states and classify them as *win* or *loss*. Next, the DB corresponding to the board states with 2 pieces (6.972 distinct states) must be constructed based on the DB for the terminal nodes and so on. The following pseudo-code resumes the algorithm for constructing the DBs (Lake et al., 1994):

1. Set all positions to *UNKNOWN*;
2. Iterate and resolve all capture positions;
3. Iterate and resolve non capture positions;
4. Go to step 3 if any non-capture position was resolved;
5. Set all remaining *UNKNOWN* to *DRAWs*.

First, every board state is classified as *unknown*. Next, the following considerations must be observed:

1. Some board states may be classified as *win* or *loss*, according to the rules of the game. For example, a player without any piece or without any legal move available is in a *loss* terminal state;

2. All state nodes which have at least one child already classified in the DB as *win* will also be classified as *win* states;
3. All state nodes whose all children have already been classified as *loss* in the DB will also be classified as *loss* states;
4. Whenever there is no more information to modify the classification of any board state in the DB, all the board states that could not be classified (that is, which remained as *unknown* states), will be classified as *draw* states.

The execution of the first iterative module of the algorithm described above (step 2) takes into account one important rule of the Draughts game: a capture move must be executed before all the other ones. Therefore, the step 2 classifies all the capture states and postpones the classification of all the remaining ones for the next steps. Since a capture which has occurred in a state with n pieces takes to a board state with (n – 1) pieces (or less), each capture state with n pieces is classified according to the BDs previously calculated. Near 50% of the states stored in the BDs correspond to capture states (Lake et al., 1994).

The execution of the second iterative module (step 3) tries to solve only the states S where no capture move is available. For each of them, all the legal moves are generated. Each legal move is executed and the values associated to the descendants of S are retrieved from the DB. The value *unknown* of S is only replaced when at least one of its children is classified as *win*, when all of them are classified as *loss*, or when all legal moves have already been solved. This procedure goes on until no board state can be solved anymore. At this point, all *unknown* states are classified as *draw* states.

In fact, there are two approaches to solve *unknown* states S:

1. **Forward Approach:** one generates the descendants of S and one tries to classify S according to them;
2. **Backward Approach:** one generates the ancestors of each solved state and one checks whether there is sufficient information to classify some of them.

The best choice depends on the proportion of solved and non solved nodes in a certain iteration process. In Chinook, it was performed a successfully combination of both approaches (Lake et al., 1994).

5.4.2. How LS-draughts uses endgame DBs

The endgame databases of Chinook that have been added to the extended LS-Draughts store information about victory, defeat or draw for:

1. All draughts board states with combination of 4 pieces x 4 pieces;
2. All draughts board states with combination of 4 pieces x 3 pieces;
3. All draughts board states involving 6 or less pieces on the board.

The main purpose of including the Chinook's endgame databases into the extended LS-Draughts is to try to answer two important questions:

1. Will the addition of the DBs into the original LS-Draughts contribute, in fact, for improving its general performance?
2. Will the use of the DBs help to decrease the rate of endgame loops in the original LS-Draughts? (This problem occurs in NeuroDraughts as well).

Next, it is presented the pseudo-code of the search algorithm of the extended LS-Draughts, followed by resuming of its main characteristics.

```
1. fun minimax (n:node, depth:int, bestmove:move) : float =
2.  if ((not isRoot(n)) and (isLookupBoard(n)))
```

```
3.      db_value := lookup_positions(n)
4.      if (db_value==1) and (n is a min node)
5.              return -1.0
6.      if (db_value==1) and (n is a max node)
7.              return +1.0
8.      if (db_value==2) and (n is a min node)
9.              return +1.0
10.     if (db_value==2) and (n is a max node)
11.             return -1.0
12.     if (db_value==3)
13.             return 0.0
14.     if leaf(n) or depth=0 return evaluate(n)
   if n is a max node
        besteval := - infinity
        for each child of n
                v := minimax (child, d-1, bestmove)
                if v > besteval
                        besteval:= v
                        thebest := bestmove
                        bestmove := thebest
        return besteval
   if n is a min node
        besteval := + infinity
        for each child of n
                v := minimax (child, d-1, bestmove)
                if v < besteval
                        besteval:= v
                        thebest := bestmove
                        bestmove := thebest
        return besteval
```

Line 2: The function *isLookupBoard()* is used to check whether the current board state *n* satisfies the constraints that allow the access to the DBs (that is, it checks whether *n* owns the adequate quantity of pieces and respects the capture constraints). The function *not isRoot()* is used to guarantee that the states *n* to be consulted in the DB have, at least, one ancestor in the search graph;
Line 3: the function *lookup_positions()* consults the DB in order to try to retrieve the prediction corresponding to *n*;
Line 4: The result *db_value = 1* indicates that *n* corresponds to a victory for the next player to execute a move and, therefore, corresponds to a defeat for its opponent (the parent of *n*). Case *n* is a *min* node, its parent is a *max* node. Therefore, the value *-1.0* must be returned in **line 5** in order to guarantee that the parent of *n* only chooses this move in case there is no other choice for it;
Line 6: The result *db_value = 1* indicates that *n* corresponds to a victory for the next player to execute a move and, therefore, corresponds to a defeat for its opponent (the parent of *n*). Case *n* is a *max* node, its parent is a *min* node. Therefore, the value *+1.0* must be returned in **line 7** as an evident prevision that the parent of *n*, normally, will not choose that move;

Line 8: The result *db_value* = *2* indicates that *n* corresponds to a defeat for the next player to execute a move and, therefore, corresponds to a victory for its opponent (the parent of *n*). Case *n* is a *min* node, its parent is a *max* node. Therefore, the value *+1.0* must be returned in **line 9** in order to guarantee that the parent of *n* chooses this move whenever it is available;
Line 10: The result *db_value* = *2* indicates that *n* corresponds to a defeat for the next player to execute a move and, therefore, corresponds to a victory for its opponent (the parent of *n*). Case *n* is a *max* node, its parent is a *min* node. Therefore, the value *-1.0* must be returned in **line 11** as an evident prevision that the parent of *n*, normally, will choose that move;
Line 12: The result *db_value* = *3* indicates that *n* corresponds to a draw for the next player to execute a move and, therefore, corresponds to a draw for its opponent (the parent of *n*). In this case, the value *0.0* is returned in **line 13**, independently of the *min* or *max* situation of *n*;
Line 14: From this line, the classical minimax algorithm (see the Minimax Search Module presented in the beginning of the section 5) is presented. Note that it will be executed only when the current state *n* is not available in the DB.

5.5 Fitness calculus

In the tournament organized to calculate the fitness of the individuals in a given generation (cited in the subsection 5.1), each individual I_i plays 10 games against the remaining individual of that generation. The results of this tournament are used to calculate the I_i fitness in the following way: 2 points for each victory, 1 point for each draw and 0 point for each defeat.

6. Endgame loop problem in the original LS-Draughts

The original LS-Draughts was executed for 30 generations with a population of 50 individuals, as presented in (Neto & Julia, 2007). For each generation, the best individual was compared with the Mark Lynch's NeuroDraughts player, in a tournament composed of 7 games, in order to evaluate its performance in relation to the latter one. The best individuals of LS-Draughts that managed to beat NeuroDraughts were the best individual I_{B-9} of the 9th generation and the best individual I_{B-25} of the 25th generation. The scores of both individuals were: 1 victory and 6 draws for I_{B-9}; 2 victories and 5 draws for I_{B-25}. Despite of the good performance of I_{B-9} and I_{B-25} against NeuroDraughts, both individuals of LS-Draughts presented endgame loop problems. As shown in the section 4, by analysing the 5 draws of I_{B-25} against NeuroDraughts, one can conclude that, in 2 of them, the LS-Draughts could have won , since I_{B-25} counted on 3 checkers and 1 simple piece on the endgame board and NeuroDraughts only counted on 1 checker. Figure 4 shows this example of endgame loop problem which affected the individual I_{B-25}: from the game board position of the figure (4.a) , resultant of the NeuroDraughts' 43rd move, the game got up to the board position of the figure (4.b) after the 44th move of NeuroDraughts. Next, the game got back to the board position indicated in figure (4.a), as a result of the 45th move of NeuroDraughts. Finally, it occurred an alternate infinite loop between these two game board positions, that is, figure (4.a) and figure (4.b), respectively.
The same problem affected the individual I_{B-9}. Analysing the 6 draws of its tournament against NeuroDraughts, 3 of them it would easily have won if I_{B-9} was able to detect endgame loops. For example, in one of these 3 games, even though I_{B-9} counted on 2

checkers and 3 simple pieces, it was not able to beat NeuroDraughts, which only counted on 1 checker and 1 simple piece.

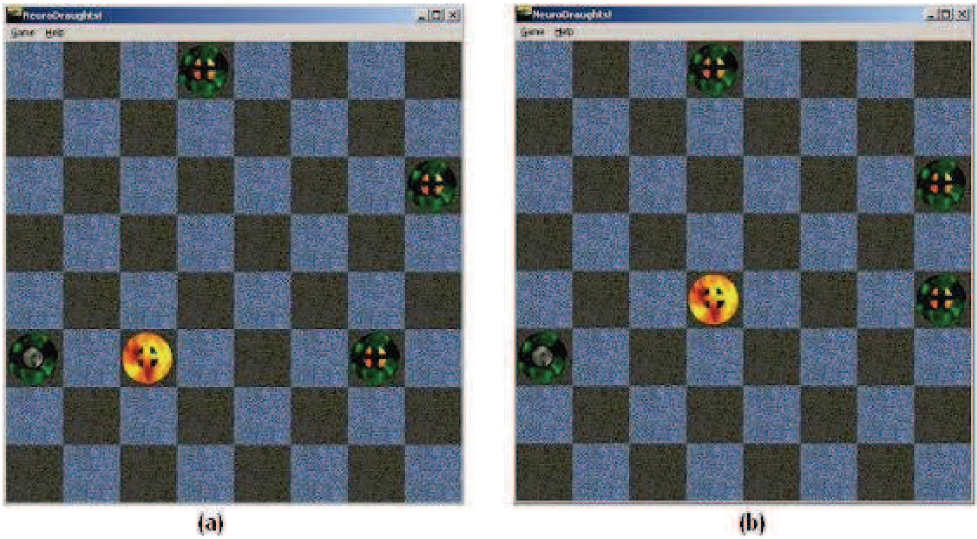


Fig. 4. Game between LS-Draughts' 25th best individual (black player) and the NeuroDraughts (red player). a) Position of the draughts game board state after the 43rd red player move; b) Position of the draughts game board state after the 44th red player move.

For many other individuals of the original LS-Draughts, the endgame loop problem came out.

7. Experimental results

The new version of LS-Draughts was executed for 50 generations with a population of 50 individuals. Differently from the original version presented in (Neto & Julia, 2007), the individuals here were trained using an endgame database module that allows to anticipate the result of the game (victory, defeat or draw) for draughts board states with, at most, 8 pieces.

As cited before, the main purpose of including the endgame databases of Chinook into the LS-Draughts was to try to answer two important questions:

1. Will the addition of the endgame databases into the original LS-Draughts contribute, in fact, for improving its general performance?
2. Will the use of the endgame databases help to decrease the rate of endgame loops in the original LS-Draughts? This problem was found by Neto and Julia in the players NeuroDraughts and original LS-Draughts (Neto & Julia, 2007).

In order to answer the first question, a tournament was executed between the available player of NeuroDraughts ($PLAYER\_1$), the best individual of the 50th generation of the original LS-Draughts ($PLAYER\_2$) and the best individual of the 50th generation of the extended version of LS-Draughts proposed here ($PLAYER\_3$):

1. FIRST MATCH: $PLAYER\_1$ x $PLAYER\_2$
   - Number of victories of $PLAYER\_2$: 5;
   - Number of defeats of $PLAYER\_2$: 1;
   - Number of draws: 8 (6 real draws and 2 draws with endgame loop problem);
2. SECOND MATCH: $PLAYER\_1$ x $PLAYER\_3$
   - Number of victories of $PLAYER\_3$: 6;
   - Number of defeats of $PLAYER\_3$: 0;
   - Number of draws: 8 (5 real draws and 3 draws with endgame loop problem);
3. THIRD MATCH: $PLAYER\_2$ x $PLAYER\_3$
   - Number of victories of $PLAYER\_3$: 3;
   - Number of defeats of $PLAYER\_3$: 1;
   - Number of draws: 10 (3 real draws and 7 draws with endgame loop problem);

Furthermore, the rates of endgame loops occurred during the training process of $PLAYER\_1$, $PLAYER\_2$ e $PLAYER\_3$ were also estimated in order to answer the second question raised above. However, to get these rates, the three players needed to be trained again, using the same training parameters, initial configuration and search, as follow:

- The player NeuroDraughts was trained using the set of features defined by Lynch in (Lynch, 1997; Lynch & Griffith, 1997), an Artificial Neural Network whose weights were generated randomly, search algorithm with depth 4 and a training strategy with 10 sessions of 200 training games. During the 2000 games, 1.045 games were finished with endgame loop problem;
- The best individual of the 50th generation of the original LS-Draughts also was trained using the following: an Artificial Neural Network whose weights were generated randomly, search algorithm with depth 4 and a training strategy with 10 sessions of 200 training games. During the 2000 games, 759 games were finished with endgame loop problem;
- Finally, the best individual of the 50th generation of the extended version of LS-Draughts was trained using an Artificial Neural Network whose weights were generated randomly, search algorithm with depth 4 and a training strategy with 10 sessions of 200 training games. During the 2000 games, 172 games were finished with endgame loop problem;

Figure 5 shows the rates of endgame loops obtained by three players during their 2000 training games. Note that que the extended version of LS-Draughts produced a rate of endgame loops corresponding to 16.46% of the one produced by NeuroDraughts, and cooresponding to 22.66% of the rate produced by the original LS-Draughts.

Games with endgame loop problems		
NeuroDraughts	**Original LS-Draughts**	**Extended LS-Draughts**
1045	759	172
100%	72.63%	16.46%
137.68%	100%	22.66%
607.56%	441.28%	100%

Fig. 5. Rates of endgame loops during 2000 training games

Theses results show that endgame DBs improved significantly the original LS-Draughts, introducing efficency and accuracy in its general performance and reducing, therefore, the rate of endgame loops.

8. Conclusion

This paper presented an extended version of LS-Draughts – a Learning System which, using endgame databases, GAs, TD methods, minimax algorithm and self-play with cloning strategy, generates a draught player much better than the original LS-Draughts. The results obtained show that:

1. The extended LS-Draughts was the best player of the tournament, with 6 victories, 8 draws and 0 defeat against NeuroDraughts and 3 victories, 10 draws and 1 defeat against the original LS-Draughts;
2. In relation to the rate of endgame loops, the extended LS-Draughts player decreased from 83% the rate of endgame loops produced by NeuroDraughts player and from 77% the rate of endgame loops produced by the original LS-Draughts player.

Therefore, the results confirm the improvement that was obtained in the original LS-Draughts with the insertion of the endgame databases module.

Nevertheless, the introduction of a search module more efficient than minimax algorithm can improve much more the performance of the extended version of LS-Draughts. That is why, as future work, the authors intend to substitute the alpha-beta algorithm combined with transposition table for the minimax algorithm of LS-Draughts.

9. References

Baxter, J. ; Tridgell, A. & Weaver, L. (1998). Knightcap: a chess program that learns by combining TD(λ) with game-tree search, *Proceedings of the 15th International Conference on Machine Learning*, pp. 28-36

Darwen, P. J. (2001). Why co-evolution beats temporal difference learning at backgammon for a linear architecture, but not a non-linear architecture, *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, IEEE Press, pp. 1003-1010

Diestel, R. (2000). *Graph Theory,* Springer-Verlag New York, Inc.

Epstein, S. (2001). *Learning to play expertly: a tutorial on hoyle,* Machines That Learn to Play Games, Nova Science Publishers, Huntington

Fogel, D. B. & Chellapilla, K. (2002). Verifying anaconda's expert rating by competing against Chinook: experiments in co-evolving a neural checkers player, *Neurocomputing*, Vol. 42, No. 1-4, pp. 69-86

Fogel, D. B. ; Hays, T. J. ; Hahn, S. L. & Quon, J. (2004). A self-learning evolutionary chess program, *Proceedings of the IEEE*, Vol. 92, No. 12, pp. 1947-1954

Gasser, R. (1990). *Applying Retrograde Analysis to Nine Men's Morris.* Available: http://citeseer.ist.psu.edu/gasser90applying.html

Gasser, R. (1996). Solving nine men's morris, *Computational Intelligence,* Vol. 12, pp. 24-41

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation,* Second Edition, Prentice Hall

Holland, J. H. (1992). *Adaptation in natural and artificial systems,* Second Edition, Cambridge, MA, USA, MIT Press

Lake, R.; Schaeffer, J.; LU, P (1994). *Solving Large Retrograde Analysis Problems Using a Network of Workstations,* Advances in Computer Chess VII, Maastricht, Netherlands, pp. 135-162

Leuski, A. (1995). *Learning of position evaluation in the game of Othello.* Available: http://people.ict.usc.edu/~leuski/publications

Levinson, R. & Weber, R. (2002). Chess neighborhoods, function combination, and reinforcement learning, *Revised Papers from the Second International Conference on Computers and Games,* Springer-Verlag, London, UK, pp. 133-150

Lynch, M. & Griffith, N. (1997). NeuroDraughts: the role of representation, search, training regime and architecture in a td draughts player, *Proceedings of Eighth Ireland Conference on Artificial Intelligence*, pp. 64-72. Available: http://iamlynch.com/nd.html

Lynch, M. (1997). *NeuroDraughts: An application of temporal difference learning to draughts.* Available: http://iamlynch.com/nd.html

Mitchell, M. & Taylor, C. E. (1999). Evolutionary Computation: An Overview, *In Annual Review of Ecology and Systematics,* Vol. 30, pp. 593-616

Neto, H. C. & Julia, R. M. S. (2007). LS-DRAUGHTS – A Draughts Learning System based on Genetic Algorithms, Neural Network and Temporal Differences, *Proceedings of 2007 IEEE Congress on Evolutionary Computation (CEC 2007)*, Singapore, ISBN: 1424413400, pp. 2523-2529

Romein, J. & Bal, H. (2002). *Awari is Solved*. Available: http://citeseer.ist.psu.edu/romein02awari.html

Romein, J. & Bal, H. (2003). *Solving the Game of Awari using Parallel Retrograde Analysis*. Available: http://citeseer.ist.psu.edu/romein03solving.html

Russell, S. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach,* Prentice Hall, Ed. 2

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development,* Vol. 3, No. 3, pp. 211-229

Schraudolph, N. N. ; Dayan, P. & Sejnowski, T. J. (2001). Learning to evaluate go positions via temporal difference methods, Computational Intelligence in Games Studies in Fuzziness and Soft Computing, Spring Verlag, Vol. 62

Schaeffer, J. ; Lake, R. ; Lu, P. & Bryant, M. (1996). CHINOOK: The world man-machine checkers champion, *AI Magazine*, Vol. 17, No. 1, pp. 21-29

Schaeffer, J. (1997). *One Jump Ahead: Challenging Human Supremacy in Checkers,* Springer-Verlag New York Inc

Schaeffer, J. ; Hlynka, M. & Jussila, V. (2001). Temporal difference learning applied to a high performance game-playing program, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 529-534

Schaeffer, J. (2002). *Applying the Experience of Building a High Performance Search Engine for One Domain to Another*

Schaeffer, J. et al. (2007). Checkers is solved, *Science*, Vol. 317, No. 5844, pp. 1518-1522

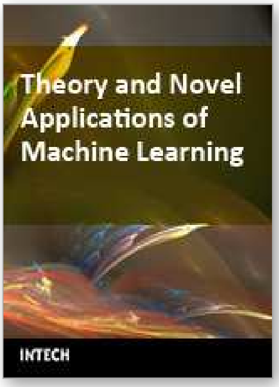
Sutton, R. S. (1988). Learning to predict by the methods of temporal differences, *Machine Learning,* Vol. 3, No. 1, pp. 9-44

Tesauro, G. J. (1992). Practical issues in temporal difference learning, *Machine Learning,* Vol. 8, pp. 257-277

Tesauro, G. J. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play, *Neural Computation,* Vol. 6, No. 2, pp. 215-219

Tesauro, G. J. (1995). Temporal difference learning and td-gammon, *Communications of the ACM,* Vol. 38, No. 3, pp. 58-68

Thrun, S. (1995). Learning to play the game of chess. *Advances in Neural Information Processing Systems 7,* The MIT Press, pp. 1069-1076



Theory and Novel Applications of Machine Learning

Edited by Meng Joo Er and Yi Zhou

ISBN 978-953-7619-55-4
Hard cover, 376 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009

Even since computers were invented, many researchers have been trying to understand how human beings learn and many interesting paradigms and approaches towards emulating human learning abilities have been proposed. The ability of learning is one of the central features of human intelligence, which makes it an important ingredient in both traditional Artificial Intelligence (AI) and emerging Cognitive Science. Machine Learning (ML) draws upon ideas from a diverse set of disciplines, including AI, Probability and Statistics, Computational Complexity, Information Theory, Psychology and Neurobiology, Control Theory and Philosophy. ML involves broad topics including Fuzzy Logic, Neural Networks (NNs), Evolutionary Algorithms (EAs), Probability and Statistics, Decision Trees, etc. Real-world applications of ML are widespread such as Pattern Recognition, Data Mining, Gaming, Bio-science, Telecommunications, Control and Robotics applications. This books reports the latest developments and futuristic trends in ML.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821