GA-Gammon

A Backgammon Player Program Based on Evolutionary Algorithms

Irineo-Fuentes, Oscar; Cruz-Cortes, Nareli; Rodriguez-Henriquez, Francisco; Ortiz-Arroyo, Daniel; Larsen, Henrik Legind

Published in:
Proceedings of the Fifth Mexican International Conference on Artificial Intelligence, (MICAI'06)

DOI (link to publication from Publisher):
10.1109/MICAI.2006.23

Publication date:
2006

Document Version
Early version, also known as pre-print

Link to publication from Aalborg University

Citation for published version (APA):
Irineo-Fuentes, O., Cruz-Cortes, N., Rodriguez-Henriquez, F., Ortiz-Arroyo, D., & Larsen, H. L. (2006). GA-Gammon: A Backgammon Player Program Based on Evolutionary Algorithms. In *Proceedings of the Fifth Mexican International Conference on Artificial Intelligence, (MICAI'06)* (pp. 207–216). IEEE Computer Society Press. https://doi.org/10.1109/MICAI.2006.23

GA-Gammon: A Backgammon Player Program Based on Evolutionary Algorithms

Oscar Irineo-Fuentes
Nareli Cruz-Cortés
Francisco Rodríguez-Henríquez
CINVESTAV-IPN
Electrical Engineering Department, Computer Science Section
Mexico
{oirineo,nareli}@computacion.cs.cinvestav.mx
francisco@cs.cinvestav.mx

Daniel Ortiz-Arroyo
and Henrik Legind Larsen
Computer Science and Engineering Department
Aalborg University Esbjerg
Denmark
{do,legind}@cs.aaue.dk

Abstract

In this paper we describe a genetic algorithm approach able to confection strong backgammon automata players. We first prepared an initial vector of weights representing a set of heuristic strategies suggested by expert human players. Then, employing a genetic algorithm approach we were able to fine tune such initial vector of weights by repeatedly testing it against Pubeval, a strong benchmark player program. The vector of weights was therefore used as an evaluation function for performing a genetic heuristic selection of the best board positions during a game. Best GA-Gammon *individuals so obtained were tested in separated 5000-game tournaments against* Pubeval *itself, and* Fuzzeval*, a fuzzy controller-based player. Our experimental results indicate that the best individuals generated by* GA-Gammon *show similar performance than* Pubeval*. Furthermore,* GA-Gammon *consistently outperforms* Fuzzeval.

1 Introduction and Related Work

The study of algorithms and heuristics that could enable computers to play board games at an expert human level has received considerable attention since 1950, when Claude E. Shannon designed a computer program to play chess [10]. Unlike chess, where a combination of pruned brute force searching approaches along with opening/closing data books has been applied successfully, backgammon requires the application of radically different strategies. Backgammon is a board game that combines luck and strategy. The use of dice in backgammon avoids applying brute force-based approaches that search for the optimal moves. Moreover, different strategies must be employed in the two phases of the game i.e.

the *contact phase* where player's checkers are intermixed and the *race phase* where there is no contact anymore.

During the last decade, researchers have proposed numerous approaches to create strong player programs in the domain of backgammon. In 1995 G. Tesauro created *TD-Gammon* [11] a backgammon program that has played at the master level with humans. This program achieved a remarkable success by learning playing strategies during self playing. *TD-Gammon* employs the temporal differences method to train an Artificial Neural Network (ANN) that is used in an evaluation function. The temporal differences training strategy basically adjusts the ANN's weights according to the results obtained during a sequence of games. *TD-Gammon* consists of two ANNs that were trained specifically for the two phases of the game. Unfortunately, being a proprietary technology, *TD-Gammon* can not be used to evaluate other approaches. However, there exists a publicly available evaluation function called *Pubeval*, which was also created by G. Tesauro. The weights included in *Pubeval*'s evaluation function, were obtained using ANNs. With *Pubeval* it is possible to create a strong backgammon player program that plays at intermediate level. *Pubeval* has become the *de facto* standard benchmark in backgammon.

Pollack et al. [6] published in 1997 a backgammon player program called *HC-Gammon*, which uses a Hill-Climbing technique to train a feed-forward neural network used to create an evaluation function. *HC-Gammon* starts with an initial champion program with all weights set to zero and proceeds by playing the current champion network against a slightly mutated challenger. When the challenger wins a game, the weights in the evaluation function are changed. Using this technique Pollack et al. concluded that Tesauro's *TD-Gammon* success was mainly due to the co-evolutionary structure of the learning task and the characteristics of the backgammon game itself, and in a minor degree to the sophisticated learning techniques employed. However, G. Tesauro refuted these observations in a later paper [12].

Sanner et al.'s [9] proposal was to develop an algorithm that emulates closely the human cognition process in the domain of backgammon. They argued that humans can not explore thousands of possible moves as computer programs do. Hence, more efficient methods must be used for learning. Based on the ACT-R theory of cognition, their program *ACT-R-Gammon* analyzes general features of backgammon that may be encountered in a winning or loosing game. Bayesian learning is employed to infer the likelihood that each of the features resulting from a move would be present in a winning game. *ACT-R-Gammon* achieved a winning rate of about 46% against *Pubeval* after playing $1,000$ games.

Azaria and Sipper [1] proposed *GP-Gammon*, a backgammon player that is created using Genetic Programming. *GP-Gammon* applies Genetic Programming to the evolution of strategies for playing backgammon. After rolling the dice, the program generates all possible next-move boards. Each individual LISP program generated receives the board configuration and returns a real number that represents the score given to that board. Then, the highest scoring board is selected for playing. *GP-Gammon* was reported as obtaining a winning rate of 58% against *Pubeval* after playing $2,000,000$ games.

Recently, a fuzzy controller-based backgammon program called *Fuzzeval*, was proposed in [2]. *Fuzzeval* grades the perceived strength of all possible valid board positions it can play, using a rule base obtained from an amateur human player. *Fuzzeval* adjusts automatically the membership functions associated with the linguistic variables employed in the rule base, by aligning them with the average values that were used in the past winning games. *Fuzzeval* achieved a winning rate of 42% against *Pubeval* after playing 100 games.

Another interesting approach is that of Qi and Sun [7]. Their work proposes to use a hybrid approach were a genetic algorithm is applied to a team of agents whose learning capabilities are based on neural networks. GA-based multi-agent reinforcement learning bidding approach called *GMARLB.* Reinforcement learning performed by the multi agent system is applied in the backgammon game domain. The general idea of this approach is that individual agents within a team should be built using two decision modules: the Q module and CQ module. A single controlling agent in the team is responsible for taking the appropriate actions at any time during a game. Using the Q module, the controlling agent selects the actions to be performed, whereas the CQ module determines whether the agent should continue working as the controller or relinquish its control to other agents. Once an agent relinquishes its control, a new agent is selected by bidding algorithms. Both, the Q and the CQ modules learning capabilities are based on neural networks. In [7]. it is reported that this approach achieved a winning rate of 56% against *Pubeval* after playing $400,000$ games. Nevertheless, an independent test recently carried out in [1] showed that the high performance reported in [7] was apparently due to measurements taken in too short tournaments (50 games). According to [1], experimental results show that the winning rate of *GMARLB* reduces to just 51.2% when confronted against *Pubeval* on 1000-game tournaments.

In this paper we propose the usage of a pure genetic algorithm strategy as opposed to the hybrid genetic algorithm/neural network approach essayed in [7]. Our algorithm produces nearly the same winning rates as that of [7] with a structure much simpler and a smaller number of training games. The genetic algorithm employed by our player, *GA-Gammon*, optimize an initial vector of weights that is used as an evaluation criteria for selecting the best possible valid move in a game. The weights are produced by the combination of diverse heuristic strategies proposed by expert human players. That initial vector is used as a seed for generating a total of twelve slightly modified vectors (individuals), which are then tested against Pubeval in 1000-game tournaments. A fraction of the best fitted individuals are selected for further processing in subsequent generations. As a result, individuals are methodically refined generation after generation. Those individuals showing winning performances above 48% are separated from the rest of the population. They are re-tested in 5000-game tournaments against Pubeval. This last step is intended for confirming the winning performance found during the evolutionary process as it has been found that this figure shows a wide range of variability from tournament to tournament. After launching an extensive search of best fitted individuals, our strategy was able to find individuals showing a performance of up to 50.0% when confronted against *Pubeval* on 5000-game tournaments.

Individuals with winning performances of 50% start appearing after an average of 20 generations using a population size of 40 individuals. Recall that each individual undergoes a 1000-game tournament against Pubeval. Therefore it is fair to say that our algorithm needs an average of $40 \times 20 \times 1000 = 800,000$ games for finding the strong players reported in Section 3.

The rest of this paper is organized as follows. Section 2 describes GA-Gammon in detail. The comparative performance results of GA-Gammon while playing against Pubeval and Fuzzeval are presented in Section 3. Finally, in Section 4 we describe future work and present some conclusions.

2 GA-Gammon an Intelligent Backgammon Player

In this work we use a Genetic Algorithm (GA) to evolve a set of weights associated to the heuristic strategies employed by human experts while playing backgammon. The vector of weights contains sixteen elements. The set of strategies employed was designed following the recommendations listed in [3, 8] by a group of expert human players. The general idea of GA-Gammon's approach is to promote those board states that are evaluated as beneficial for improving the performance of our player. The vector weights are evolved by a simple *Genetic Algorithm (GA)* with the goal of maximizing the number of victories achieved by GA-Gammon. The fitness function employed by the GA is obtained by setting the GA-generated individuals to play against *Pubeval*.

2.1 Description of the Genetic Algorithm

This section describes the main features of the GA employed in GA-Gammon.

Genetic Algorithms (GA) were proposed by J. Holland in the mid 70's [4, 5]. GAs are inspired by the Neo-Darwinian principles of adaptation and survival of the fittest individuals. Although originally proposed for improving/optimizing machine learning techniques, GAs have been more applied in the domain of optimization. GAs are population-based techniques whose main characteristic is the evolution of several potential solutions (regarded as *individuals*) at the same time. Algorithm 2.1 shows the basic skeleton of a GA [1].

From Algorithm 2.1 it can be seen that a crucial factor for achieving effectiveness when using genetic algorithms is to accurately map the variables to be optimized to *individuals*, a process known as *problem representation*. It is also important to come out with an evaluation function that allows us to establish which individuals are better fitted. Furthermore the single two main mechanisms that a genetic algorithm has for finding good solutions are the *crossover operator* and the *mutation operator*. The crossover operator defines how the parents' characteristics are transmitted to the children population. The mutation operator randomly modifies a small fraction of the children population with the purpose of introducing better individuals to the evolutionary process.

Algorithm 1 Basic Skeleton of a Genetic Algorithm

```
Require: The population size PopulationSize, number of generations N.
Ensure: Finds the "best" individuals.
1: Create an initial population of individuals of size PopulationSize.
2: for $i = 0$ to $N$ do
3:    Compute the fitness value to each individual in the population.
4:    Based on their fitness value, select the individuals to be reproduced (parents).
5:    Apply the crossover operator to the parents population to create the children.
6:    Apply the mutation operator to the children population. Children are the new population.
   end for
7: Return(Best survival individuals)
```

[1]It is noted that Algorithm 2.1 corresponds to the original version of a GA. Quite a few new operators have been proposed in the literature aiming to improve the search and/or optimization capabilities of the GAs.

In the rest of this Section we explain how the general framework of Algorithm 2.1 was applied to the problem in hand. We explain in detail the specific problem representation, fitness function, crossover and mutation operators utilized by our algorithm.

2.1.1 Representation

An individual in the population is composed of sixteen elements, i. e. a chromosome is composed of sixteen genes. Each gene represents a weight w.

Let us call each individual in the population as W, with $W = w_j$, and $j \in [1..16]$. Then an individual or chromosome is composed of sixteen genes or weights w_j.

The GA's population is composed of a set of individuals.

The weights take integer values from 1 to 100, except w_4, which is defined in the range [0..36] and w_9 in range [3..15]. In the beginning the vectors are created randomly, using values defined within the range chosen for each weight.

The individuals are represented using binary numbers.

2.1.2 The Fitness Function

Each individual i, where $i \in [1..PopulationSize]$, in the population plays $1,000$ games against *GA-Gammon*. The percentage of victories obtained is assigned as the *fitness* value for the individual i.

A game against *GA-Gammon* is played by an individual using its weight vector. After the dice are rolled, all possible valid moves that an individual can make are calculated. Each chromosome evaluates each board state assigning a score S to it. The score is calculated according to the rules presented in Tables 1 and 2. The board receiving the highest score is selected for playing.

Table 1 shows how the weights w_j are used to calculate the score S for a board state if the game is in the contact phase. The score computation in the race phase is shown in Table 2.

The entire processing steps performed by an individual while playing against Pubeval is illustrated in Figure 1.

The initial population is created following the indicated representation. Then, the GA is executed in the conventional way: Assigning the fitness value to each individual in the population (percentage of victories when playing 1000 games against pubeval). Based on the fitness value, the *selection operator* is applied in order to obtain the individuals to be reproduced, i.e. the parents. Parents are recombined by applying the *crossover operator* to obtain the children. The *mutation operator* is applied to the children. The children will be the new generation. These steps are repeated until a stop criterion is met. The complete GA's process is illustrated in Figure 2.

In our case, the GA is stopped if the best fitness value does not change through 5 generations. On average the algorithm executes 20 generations.

It was determined experimentally that better results are obtained when the elitism [2] is applied to the three best individuals, i. e. the best three individuals in the population are allowed to survive.

[2]Elitism is a mechanism that allows the best individual to survive in the next generation.

Table 1: Board score computation S for the contact phase.

Let T_k be an integer variable, with $k \in [1..10]$ then

$T_1 =$ The number of pieces' blocks that are consecutive is multiplied by w_1.
$T_2 =$ The number of positions with two or more pieces multiplied by w_2
$T_3 =$ The total number of captured pieces from the opponent multiplied by w_3.
If the probability that opponent's pieces may enter the board (normalized between 0 and 36) is less than w_4 then $T_4 = w_5$, otherwise $T_4 = 0$.
$T_5 =$ The number of positions with 2 or more pieces in the home board multiplied by w_6.
$T_6 =$ The number of pieces left behind multiplied by $-w_7$.
$T_7 =$ The number of blots in the home board multiplied by $-w_8$.
If the number of pieces in one position is bigger than w_9 then $T_8 = -w_{10}$.
If the number of blots is bigger than 3, then $T_9 = -w_{11}$.
$T_{10} = -w_{12}$ multiplied by the probability that a blot is captured.

The value for the board score S in the contact phase is obtained as $S = \Sigma T_k$

Table 2: Board score computation for the race phase.

Let S and T_k be integer variables, with $k \in [1..4]$ then

$T_1 =$ The number of pieces in the own outer board multiplied by $-w_{13}$.
$T_2 =$ The number of pieces in the opponent's outer board multiplied by $-w_{14}$.
$T_3 =$ The number of pieces in the opponent's home board multiplied by $-w_{15}$.
If there is a piece outside of the home board and it is possible to move it to position 6 in the home board, then $T_4 = -w_{16}$.

The value for the board score S in the race phase is obtained as $S = \Sigma T_k$

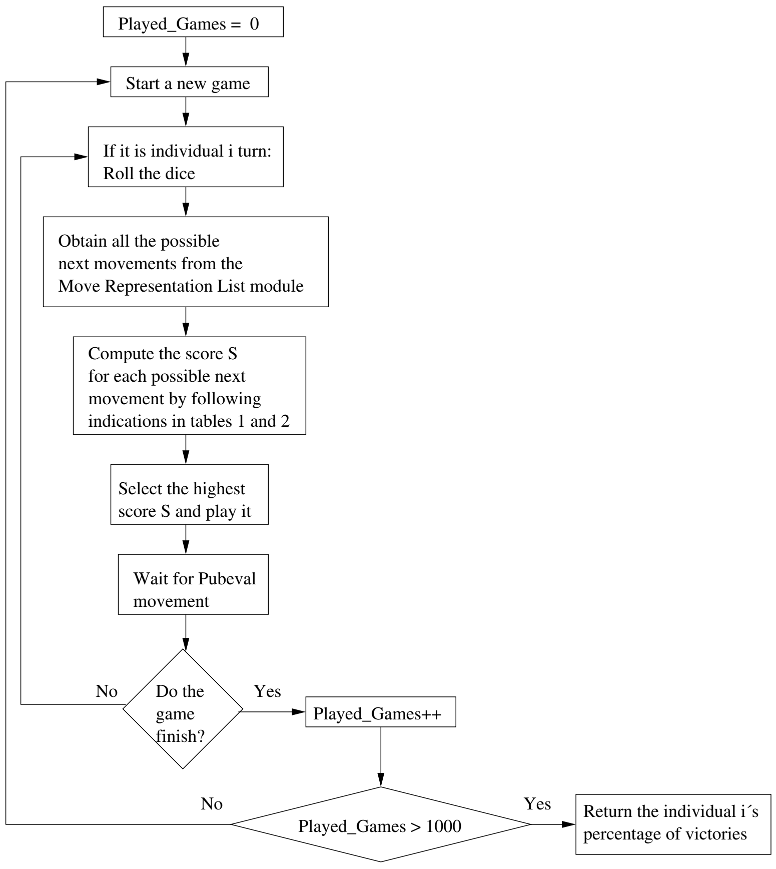

Figure 1: Processing steps performed by an individual i from the GA's population while playing against *Pubeval*.

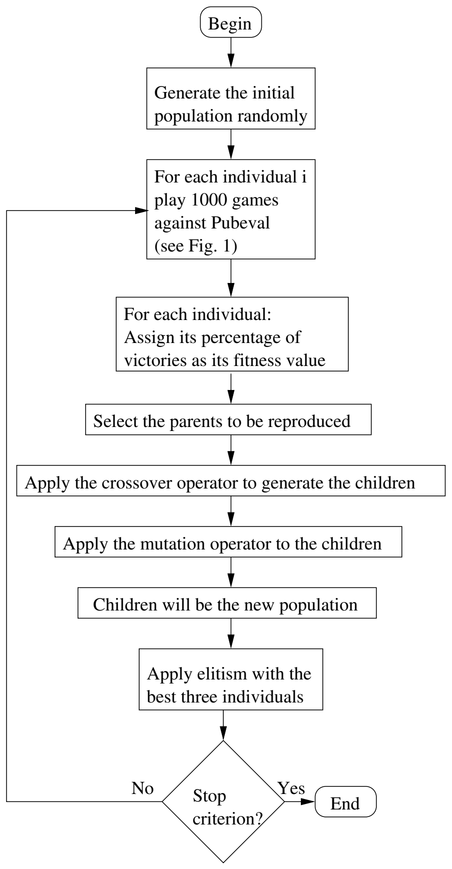
Begin
Generate the initial population randomly
For each individual i play 1000 games against Pubeval (see Fig. 1)
For each individual: Assign its percentage of victories as its fitness value
Select the parents to be reproduced
Apply the crossover operator to generate the children
Apply the mutation operator to the children
Children will be the new population
Apply elitism with the best three individuals
No
Stop criterion?
Yes
End

Figure 2: Genetic Algorithm General Structure.

3 Experimental Results

To validate the performance achieved by GA-Gammon we conducted a set of experiments, whose results are summarized in this Section. All our results were obtained by applying the following parameter values in the GA:

- Population size: 40 individuals
- Representation: Binary.
- Type of Selection: Stochastic Remainder with Replacement using Sigma scaling.
- Crossover rate: 0.9
- Mutation rate: 0.14

In order to determine the number of games that an individual should play against *Pubeval* we conducted a set of experiments. Thus, we observed that after 1000 games the variation in the winning rate is equal to 3.6% on average, after 2000 games the variation is equal to 1.2% and 0.2% after 3,000 games. Based on this information, we decided that a good compromise between winning rate and processing time would be to play 1000 games.

After we obtained the best individual from the GA process, GA-Gammon was set to play against *Pubeval* and *Fuzzeval*. The experiment consisted of playing against these players 5 sets of 1,000 games each.

Table 3: Performance comparison of GA-Gammon while playing against *Pubeval*.

	Type of Victories				
	Normal	Gammon	Backgammon	**Total**	**Percentage**
GA-Gammon	418	83	5	506	**50.0%**
Pubeval	391	101	2	494	50.0%

Table 4: Performance of GA-Gammon playing against *Fuzzeval*.

	Type of Victories				
	Normal	Gammon	Backgammon	**Total**	**Percentage**
GA-Gammon	485	78	13	576	**59.5%**
Fuzzeval	387	36	1	424	40.5

The best winning rate obtained by GA-Gammon by playing against *Pubeval* was 50.0%. The types of victories that were obtained during the tournament are shown in Table 3.

When GA-Gammon was set to play against *Fuzzeval*, the maximal winning rate obtained was 59.5%. The type of victories obtained for the best results are shown in Table 4.

In Table 5 is shown the comparison of the proposed GA-Gammon and other Backgammon players. GA-Gammon obtained very competitive results with a very simple schema.

4 Conclusions and Future Work

In this paper we have described *GA-Gammon*, a new approach to create strong backgammon player programs based on the application of evolutionary algorithms. *GA-Gammon*, employs some of the heuristic strategies that expert human players have used successfully. *GA-Gammon* employs a set of weight vectors that is considered the GA's population. Each individual in the population is set to play against *Pubeval* to determine its fitness value. The GA finds the best weight vector that maximizes the winning rate of our player.

GA-Gammon employs the information obtained from two sources of knowledge: 1) a set of heuristic playing strategies suggested by expert human players, and 2) a board score that is dependent on the percentage of victories obtained while playing against *Pubeval*.

Our experiments show that the selection of the different heuristic strategies plays an important role in *GA-Gammon*'s performance.

Table 5: Comparing GA-Gammon against other Backgammon Players

Player	%Wins vs *Pubeval*
Gp-Gammon[1]	56.8
GMARLB-Gammon [7]	51.2
GA-Gammon	**50.0**
ACT-R-Gammon [9]	45.94
Fuzzeval [2]	42.0
HC-Gammon [6]	40.00

GA-Gammon was evaluated by setting it to playing against *Pubeval* and *Fuzzeval*. Our experimental results indicate that *GA-Gammon* plays competitively with these players using a simple genetic algorithm. *GA-Gammon* obtained a winning rate of 50.0% against *Pubeval*, while clearly outperforming *Fuzzeval* with a winning rate of 59.5%.

It was shown that the proposed GA-Gammon player obtained very competitive results when compared against other players which are representatives of the state-of-the-art with a simple algorithm by training only 800,000 games in average.

In the future we will experiment with binary representation using Gray codes and different GA's parameters. Furthermore, we are planning to compute the GA-Gammon's fitness function by playing against a more powerful strategy such as TD-Gammon, instead of Pubeval.

References

[1] Y. Azaria and M. Sipper. GP-Gammon: Using Genetic Programming to Evolve Backgammon Players. In *EuroGP 2005*, volume 3447 of *Lecture Notes in Computer Science*, pages 132–142. Springer Verlag, 2005.

[2] M. Heinze, D. Ortiz-Arroyo, H. L. Larsen, and F. Rodríguez-Henríquez. Fuzzeval: A Fuzzy Controller-Based Approach in Adaptative Learning for Backgammon Game. In *Mexican International Conference on Artificial Intelligence (MICAI)*, volume 3789 of *LNAI*, pages 224–233. Springer-Verlag Berlin Heidelberg, 2005.

[3] E. Heyken and M. B. Fischer. *The Backgammon Handbook.* Crowood Press (UK), 1990.

[4] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[5] J. H. Holland. *Progress in Theoretical Biology in R. Rosen and F. M. Snell editors*, volume 4. Academic Press, 1976.

[6] J. B. Pollack, A. D. Blair, and M. Land. Coevolution of a backgammon player. In *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, pages 92–98. MIT Press, 1997.

[7] D. Qi and R. Sun. Integrating reinforcement learning, bidding and genetic algorithms. In *International Conference on Intelligent Agent Technology (IAT-2003)*, pages 53–59. IEEE Computer Society Press, Los Alamitos, CA, 2003.

[8] B. Robertie. *Backgammon For Winners, 3rd Edition.* Cardoza, 2002.

[9] S. Sanner, J. R. Anderson, C. Lebiere, and M. Lovett. Achieving efficient and cognitively plausible learning in backgammon. In *17th International Conference on Machine Learning (ICML-2000)*, pages 823–830. Morgan Kaufmann, 2000.

[10] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(314), 1950.

[11] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[12] G. Tesauro. Comments on co-evolution in the successful learning of backgammon strategy. *Machine Learning*, 32:241–243, 1998.