A Decision Making Framework for Game Playing Using Evolutionary Optimization and Learning

Alexandra Mark, Bernhard Sendhoff, Heiko Wersing

2004

Preprint:

Published in: 2004 Congress on Evolutionary Computation (CEC), vol. 1, pp. 373-380, IEEE.

A Decision Making Framework for Game Playing Using Evolutionary Optimization and Learning

Alexandra Mark
Honda Research Institute Europe
Carl-Legien Strasse 30
63073 Offenbach/M, Germany
Email: alexandra.mark@honda-ri.de

Bernhard Sendhoff
Honda Research Institute Europe
Carl-Legien Strasse 30
63073 Offenbach/M, Germany
Email: bs@honda-ri.de

Heiko Wersing
Honda Research Institute Europe
Carl-Legien Strasse 30
63073 Offenbach/M, Germany
Email: heiko.wersing@honda-ri.de

***Abstract*— We introduce a decision making framework that uses evolutionary and learning methods. It is applied to competitive games to learn online the current opponent strategy and to adapt the system counter-strategy appropriately. We compared our system for the iterated prisoner's dilemma and rock-paper-scissors with three other methods against different typical game strategies as opponents. Results show that our system performs best in most cases and is able to adapt its strategy online to the current opponent. Moreover we could show that a good prediction of the opponent is no guaranty for a good payoff, since a good prediction is often the result of a poor opponent strategy which leads to a low payoff for both players.**

I. Introduction

In competitive games several players have to make decisions about their actions. This is especially interesting when the players adapt their strategies according to the observed behavior of the opponents. To achieve this task it is sensible to build an internal model of the opponent players' strategies. There are many different methods to build such an internal model, as for example lookup-tables [1], finite automatons [2], influence diagrams [3], nested recursive models [4], or (recurrent) neural networks [5]. These internal models can be used to predict the opponents' future actions and to find suitable counter-strategies. According to [6] (p. 1593) an "anticipatory capacity is crucial for deciding between alternative courses of action". This view is supported by [7].

When humans interact, their predictions of each other are unreliable and the model which each player makes of the other depends on the model which the other one makes of oneself. In such a nested co-learning system players are often trapped by a pair of false models. For example in the iterated prisoner's dilemma (IPD) small prediction networks often over-simplify the opponent and wrongly recognize the opponent strategy as all-defection [8]. The best counter-strategy against all-defection is all-defection and thus the game will actually end with mutual defection, if the opponent correctly recognizes the all-defection strategy. Then the prediction ability of this small prediction network might misleadingly seem very good, because it correctly predicted the all-defection strategy. In our results we will also see, that the strategies with the best correct prediction rates of their opponent are not always the most successful ones.

Our original motivation was to develop a general decision making system which can solve different problems using methods like evolutionary computation and learning. These problems have to fulfill certain conditions such as a fixed number of possible actions, an existing payoff matrix, and discrete time steps, but they are not restricted to games. According to [9], humans consider different alternatives, learn and adapt their strategies, when solving decision making problems. Thus they internally play through different potential outcomes, using various models of their environment and considering different own actions. Evolutionary methods are predestinated to solve problems where a reservoir of such alternative potential solutions has to be generated. A good example for this is given in [1]. Darwen and Yao develop an evolutionary system which uses a genetic algorithm to create offline a pool of specialist strategies for game playing. These strategies are used as potential opponent-models for a hypothetical game to evaluate the outcome of different actions.

In this paper, we present a decision making system (DMS) which is applied to two different games: the iterated prisoner's dilemma and rock-paper-scissors. Our strategies are represented by neural networks. We employ a genetic algorithm to optimize the structure of the networks to allow a good strategy coding. Additionally a learning mechanism is applied to the networks in order to realize an online adaptation to the current opponent. Thus we couple evolution and learning to benefit from the advantages of both methods: The genetic algorithm generates a pool of potential network strategies which might be well suited for learning, and the learning process takes these generated nets and trains them to predict the future actions of the opponent. This pool of individuals from where the best individual is selected as "imitator" is called "imitator-pool" in the following.

In section II, we describe the structure of our DMS as well as the mechanisms it employs. Evaluation results against different strategies and the comparison with other methods are presented in section III. We conclude in section IV.

II. Method

A. Games

In this paper, the task of our decision making system is to decide on the strategy of a player in a competitive

game. The opponent is given from outside, e.g. by a fixed strategy. We have chosen the well known 2-player iterated prisoner's dilemma (abbreviated 2IPD; see e.g. [10]) and rock-paper-scissors (abbreviated RPS; also known as roshambo) as applications. In both games two players have to choose one action simultaneously in each round. In 2IPD each player has to decide whether he cooperates ("C") or defects ("D"). In the 2-person zero-sum game RPS the actions rock ("R"), paper ("P"), and scissors ("S") can be chosen. In a variant there exists one more action called well ("W"). Paper covers rock (thus the player who has chosen rock wins), rock crushes scissors (rock wins), and scissors cuts paper (scissors wins). If both players make the same choice, they tie with each other. If well is added, then well wins against rock and scissors (because they can fall into it), well loses against paper and ties against well. The corresponding payoff matrices for both games, which determine the score for each player in one round, are shown in table I. The player which plays the action given in the line of the matrix gets the given payoff against a player which plays the action denoted in the column. These payoffs are summed up over all rounds of one game. The player with the maximal sum at the end of the game has won.

TABLE I
PAYOFF MATRIX FOR IPD (LEFT) AND RPS (RIGHT)

	C	D
C	3	0
D	5	1

	R	P	S	W
R	0	-1	1	-1
P	1	0	-1	1
S	-1	1	0	-1
W	1	-1	1	0

B. Basic Algorithm

We have taken the following method by Darwen and Yao (see [1]; in the following called D/Y) as starting point for our investigations: A Genetic Algorithm (GA) is used to develop a population of IPD strategies. The last generation is kept as a diverse repertoire of specialized strategies. After the game has started, a gating algorithm searches in this repertoire for the strategies which resemble most the current opponent. The corresponding individuals are called "imitators". Then all strategies of the repertoire are tested against the selected imitators in a game over 4 rounds, starting with the current game history as input. This corresponds to a prediction of the future behavior of the opponent and a test of potential counter-strategies against it. The strategies which score best against the imitators vote for the next action to be taken in the current situation against the actual opponent.

The main differences between D/Y and our DMS are:

- Our strategies in the repertoire are represented by neural networks instead of lookup tables. This allows a more flexible coding of more sophisticated strategies.
- Our GA optimizes which part of the game history is used as input for each neural net. The GA continues online during the whole game.
- The strategies which are represented by neural nets learn online to imitate the actual opponent.
- We do not search in the strategy repertoire for potential counter-strategies against the current opponent, but we directly test all possible actions against the selected imitator. This assures that the optimal counter-action can be found, when the prediction by the imitator is correct.

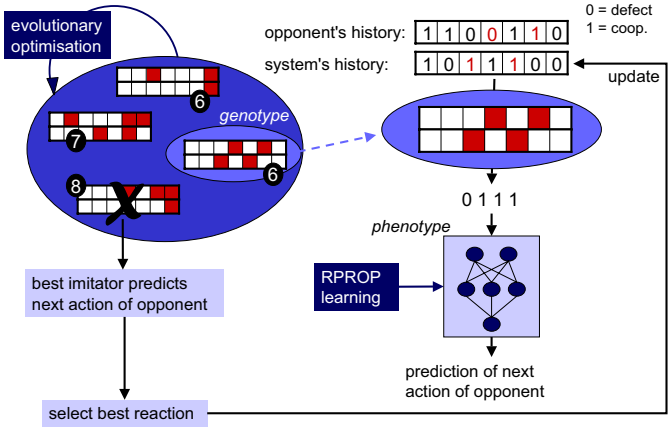


Fig. 1. Decision Making System (DMS)

Our basic algorithm is organized as follows:

1) create imitator-pool with random chromosomes
2) create a neural net for each imitator with random weights
3) repeat until end of this game (given number of rounds)
   a) compute system action
      i) in first few rounds randomly
      ii) else
         A) select from imitator-pool best imitator of opponent in past 16 rounds of game
         B) use selected imitator to predict opponent behavior 4 rounds into the future
         C) test all possible actions against the predicted opponent actions for 4 rounds into the future and select the best reaction as the next action of the DMS
   b) get action of opponent
   c) compute payoff for this round
   d) update histories
   e) all *gatRounds* rounds: optimize imitator-pool by GA and RPROP-learning

gatRounds is a random number between a given minimum (default: 3 rounds) and maximum (default: 5 rounds) which determines after how many rounds the system learns.

Figure 1 visualizes the important steps of the algorithm. The details are explained in the following sections.

C. Genetic Algorithm

The GA works on the chromosomes of the individuals in the imitator-pool, which are described in section II-D.
One GA-iteration works as follows:

1) delete worst $popSize/2$ individuals from parents; other $popSize/2$ parents are kept
2) to keep population size constant, $popSize/2$ offspring individuals are created in the following way
   a) one offspring gets random chromosome
   b) create other $popSize/2 - 1$ offspring chromosomes:
      i) with 60% probability in the following way: select one parent fitness-proportionally and the other parent randomly; then offspring is created by 2-point-crossover
      ii) with 40% probability in the following way: select one parent fitness proportionally and use it directly as offspring
3) for each offspring
   a) mutate chromosome (flip probability 20%)
   b) create neural net with random weights (number of input neurons is given in corresponding chromosome)
   c) each neural net learns with RPROP algorithm from previous game history to imitate opponent (some learn parameters are given in corresponding chromosome)
4) add new offspring to new parents population
5) fitness computation of population as described in section II-F. Considered are
   a) the (possibly weighted) number of matches with the opponent actions in the previous rounds
   b) if the individual has already given a correct imitator-prediction once before
   c) a small penalty for each used history window

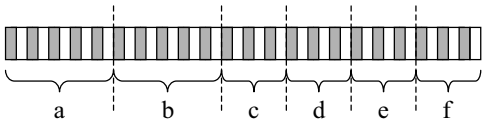
In the experiments shown at the end of this paper we used a population size of 50. In each optimization step (see section II-B) 10 generations of the genetic algorithm are executed and the population is stored as a new optimized imitator-pool. The learning of the corresponding neural nets is described in section II-E.

D. Strategy Representation

In each round the actions of both players are stored in a game history – one for each player. The actions are coded as consecutive integers: "0" = defect and "1" = cooperate for IPD, and "0" = rock, "1" = paper, "2" = scissors, "3" = well for RPS. The index of the history vector denotes the round of the game.

Each individual (or strategy) is represented by a neural network with its weights and a chromosome which codes additional parameters. The nets are fully connected feed forward neural nets with one hidden layer of 10 neurons. Each net gets as input a part of the current game history and gives as output an action. The GA optimizes which part of the history is used as input (see below), thus the number of input neurons of the nets differ.

The binary chromosome codes several parameters (see figure 2). The first 10 bits code the windows to look on the system



with
a: windows for own history
b: windows for opponent's history
c: number of learn steps from learn history (*numLearnSteps1*)
d: number of learn steps from game history (*numLearnSteps2*)
e: number of learn rounds (*maxLearnRounds*)
f: further learning parameter (*learnBackHist*)

Fig. 2. Coding of the binary chromosome

history. If the bit is "1", then the corresponding position of the history can be used, but not if the bit is "0". The first bit corresponds to the oldest part of the history (the action before 10 rounds) and the 10-th bit corresponds to the most recent action. The same applies for the next 10 bits which code the windows for the history of the opponent (see figure 3). "10" is the default value of a parameter which determines how many windows can be maximally used.

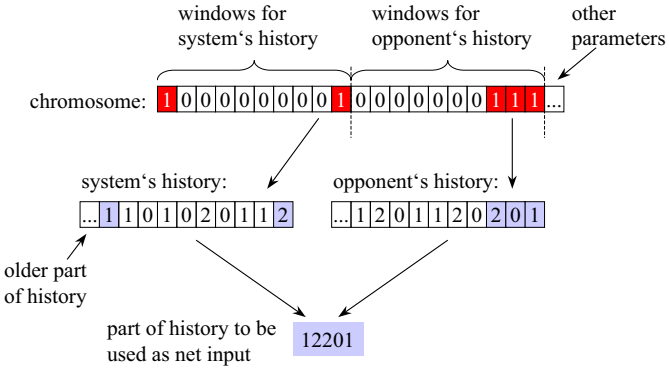


Fig. 3. History windows code which parts of the history are used as input for the neural net that predicts the next action of the opponent

The next four values (c–f in figure 2) are integer values. *numLearnSteps1*, *numLearnSteps2*, and *maxLearnRounds* lie between 0 and 50, whereas *learnBackHist* can take values between 0 and 100. These four parameters are needed for the learning process and are described in section II-E. For each value 6 bits are Gray coded and rounded to an integer value in the given interval. Thus the length of the chromosome is $2 \cdot 10 + 4 \cdot 6 = 44$.

The length of the lookup-table-type chromosome of [1] increases fast with a rising memory length of the coded strategies. For a memory length (m) of 4 the chromosome length is already 264 ($2^{2m} + 2m$). This is the longest memory length which has been considered in Darwen's and Yao's paper. Since we are using neural nets in addition to the chromosome to code our strategies, our chromosomes are much shorter and independent of the memory length.

E. Learning

We use fully connected feed forward neural nets with one hidden layer of 10 neurons. Since the GA optimizes which part of the history is used as input, the number of input neurons depends on the number of history windows (h) which are coded in the chromosome. Let *numActions* denote the number of possible actions for each player. Thus the input (which is a part of the game history) consists of h integer values between 0 and $numActions - 1$. Each input value is 0-1-coded and given into $numActions$ input neurons in the following way: For input value 0 the first neuron gets input 1 and the neurons 2 until numActions get input 0. For input value 1, the second neuron gets input 1, and the others get 0. See also figure 4 for clarity. Thus the number of input neurons is computed as $h \cdot numActions$.

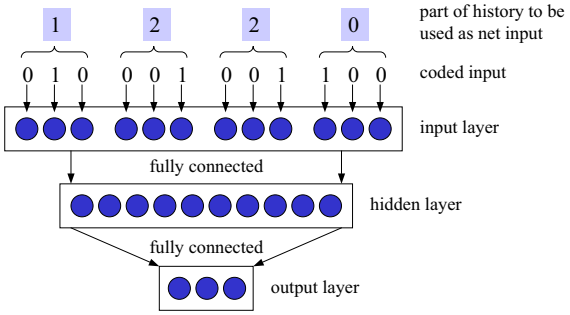


Fig. 4. Neural network with example input coded for $numActions = 3$

The input neurons propagate the given input directly to the next layer. The hidden neurons and the output neurons have a sigmoid activation function $z_i = 1/(1 + exp(-a))$, with a being the propagated result of the previous neurons: $a = \sum_j w_{ij} z_j + \Theta_i$. Θ_i is a bias term between -1 and 1. w_{ij} is the real-valued weight $\in [-1, 1]$ of the connection from neuron j to neuron i. When a new net is created, all weights and bias values are initialized with uniformly distributed random values $\in [-0.1, 0.1]$.

Each network has *numActions* output neurons – one for each possible action. The action corresponding to the neuron with the maximal output value determines the action of the corresponding individual (winner takes all). If during the game a net should be used to compute an action, but the available game history is shorter than needed according to the history windows, a random action is returned instead.

The RPROP (resilient backpropagation, see [11]) learning rule is applied as learning mechanism to adapt the real-valued weights. The learning data are taken from the past game history: One random position between the current position and up to *learnBackHist* rounds back into the past is determined in the history. The target output for the net is the action of the opponent in the round after the selected random position. Thus the input for the learning net is read from the history (with the random position as newest available history data) according to the history windows which are coded in the corresponding chromosome, and the error is computed using the difference between net output and target. The parameter *learnBackHist* is coded in the chromosome, as well as the number of learn steps *numLearnSteps2* for each learning process and the maximal number of rounds of the game in which the net can learn (*maxLearnRounds*). Thus all *gatRounds* rounds each imitator net does *numLearnSteps2* steps of RPROP learning until the total number of learn rounds (*maxLearnRounds*) is reached. Then the weights of the net are "frozen". This allows to keep strategies as they are in the pool, because they are not changed any more by further learning.

numLearnSteps1 was intended to determine the number of steps to be learned from an artificially created history which should be given for initial learning before the start of the real game. The idea was to provide with this learn history some typical strategy mixture. However, it became clear very soon, that the online learning is so fast, that no beforehand learning is needed. The results showed no better performance when the offline learning was added. Thus in the experiments which are described at the end of this paper, the value of *numLearnSteps1* was ignored and no offline learning was done.

F. Imitator Selection

In each round of the game the best imitator is selected from the imitator-pool in order to predict the next action of the opponent. The following items are considered to find the best imitator. The values add up to a reward, which is also used as fitness for the genetic algorithm.

- A parameter (*numPastRounds*, default: 16) specifies the number of previous rounds of the game history in which each individual is compared with the real opponent. For each round in which an individual selects the same action as did the opponent in this situation, the individual gets 1 point. Here the individual gets the same history as input which was available to the opponent at this time step. The points which the individuals get for matching the opponent behavior can be weighted, so that matching more previous rounds gets more reward than matching older parts of the history. For the experiments which are described at the end of this paper however, this weighting has not been used.
- Each individual which has been selected once as imitator and has predicted the correct action, gets a positive additive value (+1), which contributes to the reward (r), but decays slightly with each round *t* of the game. Thus:

$$r(t+1) = \begin{cases} (r(t)+1) \cdot decay & , \text{ if correct pred.} \\ r(t) \cdot decay & , \text{ else} \end{cases}$$

  with *decay* being a parameter between 0 and 1 (default: 0.95).
- For each used window position in the history the reward is reduced by a small value (default: 0.2).

The individual with the highest reward is selected as imitator. When several individuals have the maximal sum, the imitator is selected randomly among these.

G. Reaction to Imitator Action

The following mechanism for finding the best reaction to the predicted opponent action has been implemented: All possible actions which can be played starting with the current game history are tested for the next *numNext* rounds into the future in a hypothetical game. Here the part of the opponent is simulated using the imitator and the system player tests all possible actions it could take. The system action of the first tested round which led to the maximal payoff for the system after all *numNext* rounds (i.e. highest utility for the system) is taken as the next action of the system in the actual game (see figure 5).

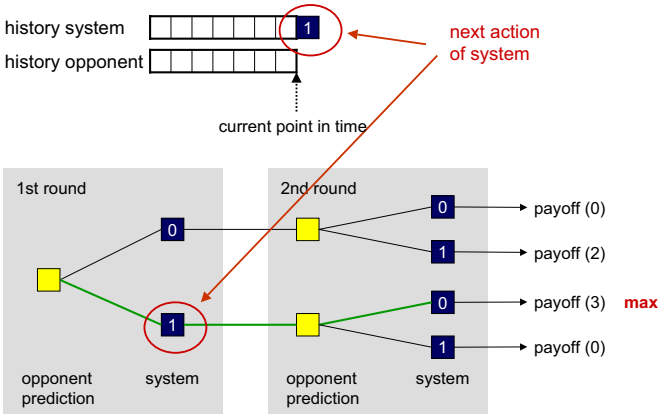


Fig. 5. Action selection tree

Obviously this method can only be employed in those cases where the number of possible actions is rather small and all are known. Thus this approach has to be enhanced in order to work with more general problems. The same method, to use a model to predict the future actions of the opponent, to test all possible actions against this model for some steps into the future and to select the first system action of the best path, is also employed in [5].

In the implementation [1] all individual strategies in the population from where the imitators are selected also serve as the pool from where the answer strategies are selected. Thus having the imitators, all individuals of this pool are tested against the imitators. The test game is played for every two players over 4 rounds into the future, starting with the current game history. Those individuals which score best against the imitators participate in a majority voting to determine the final action. This method has the drawback, that the reaction-strategies are limited to a small pool of given strategies. So a good prediction of the next actions of the opponent can possibly not be exploited to full extend, because no suitable counter-strategy is available. In [1] on page 104 Darwen and Yao propose another time-consuming method, which they did not try: “One could run another GA for each individual to search for the single best counter-strategy for that individual”.

Darwen and Yao used the value 4 for the parameter *numNext*, because this ”gave the best performance after a modest amount of searching” (see footnote 5 on page 105 in [1]). We would like to add a theoretical justification for, respectively against, some particular values: According to [10] the payoff matrix which we have used as well as the payoff matrix which has been used in [1] fulfill the conditions of a pure PD where universal cooperation is Pareto optimal, especially: $R \geq (T+S)/2$. Here R means reward (the payoff, when both players cooperate), T stands for temptation (if one defects alone), and S is the sucker payoff (if one cooperates and the opponent defects). If both players defect, the payoff is called punishment or P. For uneven values of *numNext*, alternating between T and S might get a higher sum over all *numNext* rounds than only playing R. With *numNext* = 3, the sequence TST gets a summed payoff of 10, whereas RRR (universal cooperation) only gets 9 points (see table I). Because the overall strategy of the decision system depends strongly on the maximal summed payoff over the *numNext* rounds (see section II-G), these cases might mislead the system to defect more often (get more T) than it would be sensible. Thus one should pay attention to only use values for *numNext* which fulfill the following inequality (to assure that universal cooperation always gets a higher summed payoff than alternating T and S):

$$\frac{numNext+1}{2} \cdot T + \frac{numNext-1}{2} \cdot S \leq numNext \cdot R$$

III. Results

For evaluation we compared our DMS with three other methods: “D/Y”, “linear prediction”, and “single neural net”. D/Y is our re-implementation of the system described in [1]. The two implementations might still differ in some details or parameter values. Darwen and Yao used a different payoff matrix, but we decided to take the matrix which is used at the CEC’04 PD competition. Nevertheless the main features of their system have been covered. In the linear prediction method, we used a classical linear predictor of order 3 to predict the next action of the opponent, with the game histories of both players as input. In this case the actions have been rounded to the allowed values. In the single neural net method we used a neural net similar to those in the DMS, but with a fixed number of input neurons: the previous 5 actions of both players’ histories are fed into the net which learns online, using the RPROP-algorithm (as in the DMS), to predict the next action of the opponent. For the latter two methods, the counter-action is computed in the same way as in the DMS (see figure 5), but instead of an imitator prediction, the prediction of the single available linear predictor respectively neural net is used.

There is a PD competition organized at CEC’04 celebrating the 20th anniversary of the famous Axelrod competition. On the corresponding web site (http://www.prisoners-dilemma.com) eight default strategies are given. For evaluation, we had our DMS play 10 runs of 1000 rounds against each of those given strategies:

RAND makes random moves
ALLD always defects

ALLC always cooperates

GRIM starts with cooperation and plays ALLD after first defection of opponent

TFT tit-for-tat starts with cooperation and repeats last move of opponent

STFT suspicious TFT: like TFT but defects in first move

TFTT tit-for-two-tats: like TFT but forgives one defection, i.e. defects only after two consecutive opponent defections.

Pavlov starts with random move and repeats its last move, if it has brought many points (two higher payoff values in table I), otherwise makes other move

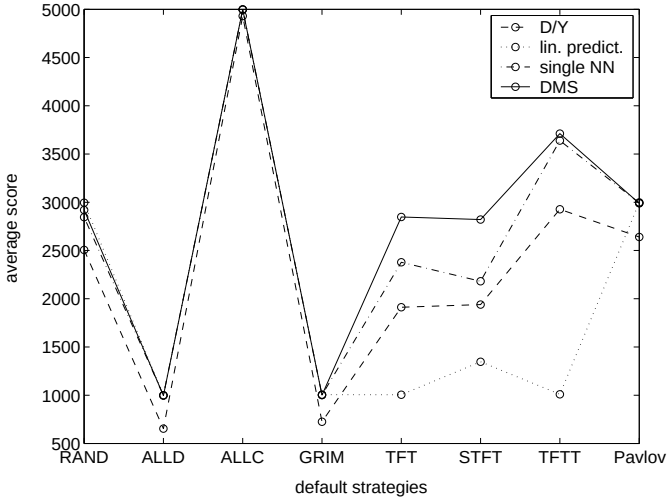


Fig. 6. Average score of different methods against CEC'04 PD default strategies (10 runs of 1000 rounds for each method; payoff matrix see table I). Standard deviations are given in table II.

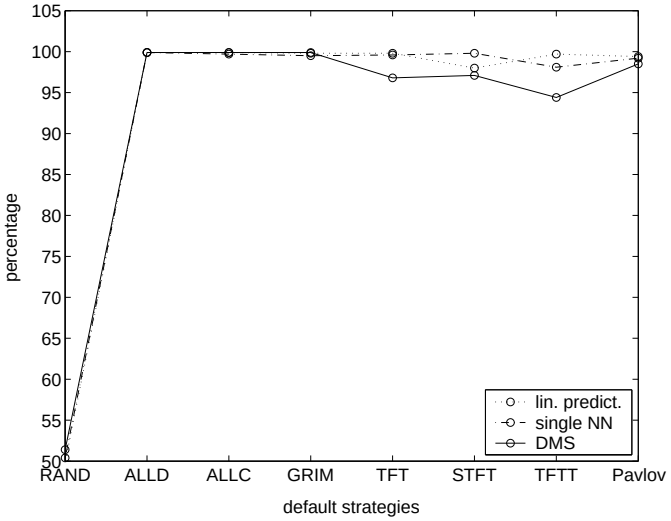


Fig. 7. Percentage of correct opponent predictions by different methods against CEC'04 PD default strategies (10 runs of 1000 rounds for each method).

The DMS gets sensible results against all those strategies: It plays (nearly) only defection against RAND, ALLD, ALLC, and GRIM. This is the best, it can do, when not knowing the opponent strategy but having to explore it online during the game. Against TFT and STFT mostly cooperation is played. Against TFTT after an initial phase of mostly defection, the DMS repeats most time the pattern "1010". Against Pavlov the DMS either played many rounds of defection or some irregular pattern of "0" and "1". Both strategies get about the same payoff at the end.

Figure 6 and 7 show the results. In figure 7 the percentage of correct predictions of the next opponent action is plotted. The D/Y method is not included here, because no explicit prediction is used. They select a number of imitator strategies instead, which are used to find a counter-action.

TABLE II
DIFFERENT METHODS AGAINST CEC'04 PD DEFAULT STRATEGIES

		D/Y	lin. pred.	single NN	DMS
RAND	avg	2505.4	2995.4	2845.7	2919.9
	std	104.8	58.2	121.4	51.1
	max	2726	3114	3035	3004
	min	2326	2936	2591	2814
	pred	—	49.8	50.4%	51.4%
ALLD	avg	653.3	999.3	998.3	996.7
	std	133.8	0.5	1.1	2.1
	max	841	1000	1000	1000
	min	507	999	996	997
	pred	—	99.9%	99.9%	99.9%
ALLC	avg	4930.4	4998.6	4996.6	4998.8
	std	209.7	1.0	1.9	1.4
	max	5000	5000	4998	5000
	min	4334	4998	4994	4996
	pred	—	99.9%	99.7%	99.9%
GRIM	avg	724.8	1005.6	1002.0	1001.6
	std	249.5	0.8	2.1	1.8
	max	934	1006	1004	1004
	min	258	1004	999	1000
	pred	—	99.8%	99.5%	99.9%
TFT	avg	1911.7	1005.2	2377.7	2847.3
	std	285.6	1.0	948.8	74.2
	max	2423	1006	2996	2886
	min	1567	1004	1004	2644
	pred	—	99.8%	99.6%	96.8%
STFT	avg	1939.5	1346.3	2181.2	2820.3
	std	358.3	724.8	1017.1	180.9
	max	2487	2722	2994	2886
	min	1285	1000	1000	2306
	pred	—	98.0%	99.8%	97.1%
TFTT	avg	2927.1	1009.0	3639.7	3711.2
	std	783.5	1.0	921.4	44.0
	max	3991	1010	3979	3744
	min	1571	1008	1020	2306
	pred	—	99.7%	98.1%	94.4%
Pavlov	avg	2639.8	2998.4	2992.1	2988.9
	std	448.1	4.7	14.2	8.9
	max	3002	3002	3000	3000
	min	1849	2989	2953	2976
	pred	—	99.4%	99.2%	98.5%

Table II shows average (avg), minimal (min), and maximal (max) payoffs as well as standard deviations (std) and percentages of correct opponent predictions (pred) of the different methods. Compared to the other methods, our DMS belongs always to the best methods and often gets *the* best score. Against the simple strategies ALLD, ALLC, and GRIM, the

DMS has no chance to outperform the other strategies at large. The DMS plays much better than the other methods against TFT and STFT. Whereas all other methods get low minimal values, the games of the DMS against TFT and STFT have never ended with mutual defection, which has led to the low scores of the other methods. The very high standard deviations of many methods against TFT and variants occur because some of the games have converged to mutual cooperation (high score), and some to mutual defection (low score).

The D/Y method shows a relatively high standard deviation against all default strategies. This can probably be explained by the method itself: the strategy reservoir is created offline before the game starts, optimized by a GA which uses for evaluation only games between strategies of the reservoir itself. Thus the population might converge to very special strategies, e.g. rather cooperative ones, which cannot play well against all unseen test strategies. In [1] Darwen and Yao claim that their method generalizes well and performs well against unseen test strategies, but they do not measure the diversity of their created reservoir to prove this.

In the IPD results shown in table II, our DMS was better than the single learning neural net on the average over 10 runs. But against the TFT variants the single learning neural net could reach a higher maximum payoff. Closer analysis of these cases with the single neural net method shows for TFT and STFT that mutual cooperation is reached very fast within the first few rounds and is played until the end of the game. For TFTT the optimal pattern which alternates between 0 and 1 is reached within the first 100 rounds and is then played up to the end of the game. The DMS however does not play the optimal patterns up to the end, once they have been found, but tries some defective moves in between every now and then. Thus the maximal payoff of the DMS is slightly lower than the maximal payoff of the single neural net method, but the DMS gets a better average value, because it never ends with mutual defection. In this respect the DMS is more robust. According to [8] humans' prediction learning does not converge as fast as typical neural network learning, thus allowing more exploration. In our experiments the DMS also behaved more explorative than the single neural net and did not converge as fast. When playing RPS with its four possible actions, the DMS beats the neural net method clearly.

In order to show the flexibility of our approach, we also investigated the performance of our DMS in the game RPS. Similar to the famous Axelrod IPD competitions, there have also been several public RPS competitions. The winning strategy of the First International RoShamBo Programming Competition 1999 ([12]) is called "iocaine powder" (see description and source code at [13]). This strategy consists of several sub-strategies and meta-strategies including different prediction methods applied over multiple time scales. For evaluation, we had the DMS play 10 games of 1000 rounds of RPS (with the actions rock, paper, and scissors, but not well). As a result, the DMS won 2 out of 10 games, reached as average payoff -14, maximal payoff 54 and standard deviation 33.2. Since RPS is a zero-sum-game, two equally good players would get payoff 0 at the end. Thus the DMS plays better than expected against this very specialized game strategy. Our DMS however has already been applied to two different games and will be applied to other decision problems which do not belong to the area of game playing. The percentage of correct predictions of the iocaine powder strategy by the DMS is 33% – the same as guessing with three possible actions. There is also no special pattern visible in the history of both players. But this has to be expected when two good strategies play RPS against each other, since random play cannot be beaten at large. High payoffs are only possible if one strategy recognizes the weakness of its opponent – e.g. in form of an easily predictable repetitive pattern of actions – and adapts its own strategy accordingly in order to exploit this. But obviously neither iocaine powder nor DMS have such weaknesses.

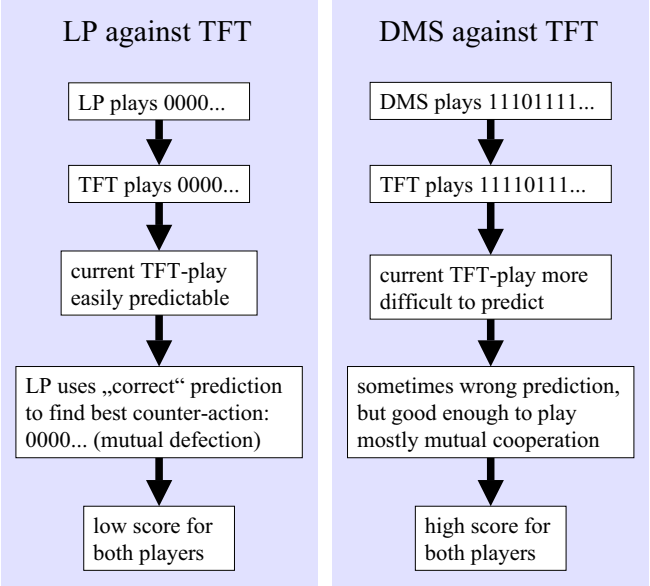


Fig. 8. Example of PD where better prediction leads to lower score

At first sight it might be surprising that a better prediction does not always give rise to a better score (see figure 6 and 7). But this can easily be explained, because simple strategies are easy to predict but often lead to a low score for both players. Example (cf. figure 8): The linear predictor (LP) only defects against TFT. Therefore, TFT also defects and is easily predictable, but the score for both players is low. The DMS plays mostly cooperation against TFT in a more sophisticated pattern (one defective action every now and then between cooperative actions). Therefore, also TFT mostly cooperates using this pattern which is not so easily predicted as continuing defection. Thus the prediction is worse but the score is higher than in the former case. This does not mean that a good prediction of the opponent actions is not always translated into good counter-actions of the system. The good prediction is only the result of a bad action selection which leads to a very homogeneous game history which can be easily predicted. In figure 8 the linear predictor wrongly recognizes the TFT strategy as ALLD, thus it only seems to be a good prediction

because nearly 100% of correct predicted actions are reached throughout the game.

A similar phenomenon can be seen in table III where four sophisticated methods played IPD or RPS against the DMS. All IPD-games except D/Y converged very fast to mutual defection, thus the actions of the players are predicted correctly with very high percentages (about 99%), but the result is a very low score. When the DMS played against D/Y, then typically the DMS defected nearly always and the D/Y player played in some games a very regular pattern, such as "110000110000..." or "10001000...", but sometimes also a more irregular pattern or mostly defection. The average payoff for the DMS was much higher than for D/Y. The DMS won all games and predicted 85.9% of all D/Y actions correctly. Thus here again the lower correct prediction rate of the DMS against D/Y lead to a higher payoff than the higher correct prediction rate of the DMS against the other methods.

TABLE III

RESULTS OF DMS AGAINST DIFFERENT METHODS; IN EACH CELL FIRST GIVEN VALUE BELONGS TO DMS, SECOND TO OPPONENT METHOD

		D/Y	lin. pred.	single NN	DMS
DMS IPD	avg	2081.3 817.3	1001 1013	1000.1 1019.6	1013.8 1011.3
	std	575.4 817.3	5.8 6.1	6.8 8.4	7.7 5.8
	max	2773 1000	1015 1022	1018 1033	1024 1022
	min	1206 593	996 1000	994 1008	1002 1006
	pred	85.9% —	99.7% 98.9%	99.6% 99.2%	99.3% 99.3%
DMS RPS	avg	— —	26.3 -26.3	86.3 -86.3	2.7 -2.7
	std	— —	23.7 23.7	34.6 34.6	26.1 26.1
	max	— —	55 14	143 -42	42 33
	min	— —	-14 -55	42 -143	-33 -42
	pred	— —	33.5% 24.3%	32.1% 25.2%	28.8% 28.4%

When the same methods play RPS (with well) against each other, their actions become almost unpredictable and the rate of correct predictions sinks to values in the range of 30%. When playing RPS against itself, the history of the DMS shows no recognizable special pattern and the game ends in about the same payoff for both players. Since the D/Y method only works for IPD (because of the strategy coding), we could not compare it with the other methods for RPS.

IV. CONCLUSION

The DMS outlined in this paper beats the single neural net in RPS and plays equally good in IPD (mutual defection). Moreover the average payoff against the CEC default strategies in IPD is higher than for the single neural net. Since both methods use online learning, one can conclude, that the usage of a *pool* of neural net strategies and the GA optimization of the history windows result in the observed advantage.

The DMS beats the D/Y-method in IPD and plays better against the CEC default strategies. Since both methods use a strategy pool, one can conclude, that our enhancements like online learning, strategy coding with neural nets instead of lookup tables, GA optimization of history windows and explicitly trying out all possible counter-strategies must have brought the advantage. It is planned to analyze in more detail which relative influence is due to which modification.

Despite online-learning, the DMS runs faster than our implementation of the D/Y method (including D/Y offline pool generation). Especially the disassortative sampling in the D/Y method is very time-consuming. Since Darwen and Yao did not measure the diversity or show in [1] how much better their system performs using fitness-sharing and disassortative sampling than without these methods, we did not employ these methods for the DMS.

As intended, our DMS learns online the current strategy of its opponent (for RPS or IPD) and adapts its own strategy accordingly. The performance is very good compared to other playing methods, as has been shown in section III. Since the DMS is not primarily intended to play games, we want to apply it to other applications concerning decision making problems, in order to fully exploit its potential.

ACKNOWLEDGMENT

This work was supported by the German Ministry of Education and Research (BMBF) under grant LOKI 01IB001E. We would like to thank Edgar Körner for his support, Paul Darwen for answering several questions concerning the parameters used in his paper [1], and Georg Schneider for his assistance with the linear prediction method.

REFERENCES

[1] P. J. Darwen and X. Yao. Speciation as automatic categorical modularization. *IEEE Trans. Evolutionary Computation*, 1(2):101–108, 1997.

[2] D. Carmel and S. Markovitch. Learning models of intelligent agents. In *Proc. 13th Nat. Conf. AI*, pages 62–67. AAAI Press, 1996.

[3] D. Suryadi and P. J. Gmytrasiewicz. Learning models of other agents using influence diagrams. In *Proc. Int. Conf. User Modeling*, pages 223–232, 1999.

[4] J. M. Vidal and E. H. Durfee. Recursive agent modeling using limited rationality. In *Proc. 1st Int. Conf. on Multiagent Systems (ICMAS)*, 1995.

[5] M. Taiji and T. Ikegami. Dynamics of internal models in game players. *Physica D*, 134(2):253–266, 1999.

[6] W. Schultz, P. Dayan, and R. P. Montague. A neural substrate of prediction and reward. *Science*, 275:1593–1599, 1997.

[7] M. V. Butz, O. Sigaud, and P. Gérard, editors. *Anticipatory behavior in adaptive learning systems*. Springer, New York, 2003.

[8] T. Ikegami and G. Morimoto. Chaotic itinerancy in coupled dynamical recognizers. *Chaos*, 13(3):1133–1147, September 2003.

[9] J. W. Payne, J. R. Bettman, and E. J. Johnson. *The adaptive decision maker*. Cambridge University Press, 1993.

[10] S. Kuhn. Prisoner's dilemma, 2003. The Stanford Encyclopedia of Philosophy (Fall 2003 Edition), http://plato.stanford.edu/archives/fall2003/entries/prisoner-dilemma/.

[11] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. IEEE Int. Conf. Neural Networks*, volume I, pages 586–591. IEEE/INNS, 1993.

[12] D. Billing. The first international roshambo programming competition, 1999. http://www.cs.ualberta.ca/ darse/rsbpc1.html.

[13] D. Egnor. Iocaine powder, 1999. RPS strategy, http://dan.egnor.name/iocaine.html.