Expected Payoff Analysis of Dynamic Mixed Strategies in an Adversarial Domain

Pablo J. Villacorta and David A. Pelta
Models of Decision and Optimization Research Group
CITIC-UGR, Department of Computer Science and AI, University of Granada
C/ Periodista Daniel Saucedo, 18071 Granada, Spain
Telephone: (+0034) 958 24 23 76
Email: {pjvi, dpelta}@decsai.ugr.es

***Abstract*—Adversarial decision making is aimed at determining optimal decision strategies to deal with an adversarial and adaptive opponent. One defense against this adversary is to make decisions that are intended to confuse him, although our rewards can be diminished. In this contribution, we describe ongoing research in the design of time varying decision strategies for a a simple adversarial model. The strategies obtained are compared against static strategies from a theoretical and empirical point of view. The results show encouraging improvements that open new venues for research.**

***Index Terms*—Adversarial reasoning, decision strategies, decision making**

I. Introduction

Adversarial decision is largely about understanding the minds and actions of ones opponent. It is relevant to a broad range of problems where the actors are actively and consciously contesting at least some of each others objectives and actions [1]. The field is also known as decision making in the presence of adversaries or adversarial reasoning.

In its most basic form, adversarial decision making involves two participants, S and T, each of which chooses an action to respond to a given event without knowing the choice of the other. As a result of these choices, a payoff is assigned to the participants. When this scenario is repeated many times, i.e. situations of repeated conflicting encounters arise, then the situation becomes complex as the participants have the possibility to learn the others strategy. Examples of this type can be found in the military field, but also in problems of real-time strategy games, government vs government conflicts, economic adversarial domains, team sports (e.g., RoboCup), competitions (e.g., Poker), etc. [1]

Adversarial decision making is aimed at determining optimal strategies (for S) against an adversarial and adaptive opponent (T). One defense against this adversary is to make decisions that are intended to confuse him, although S's rewards can be diminished. It is assumed that making decisions in an uncertain environment is a hard task. However, this situation is of upmost interest in the case of adversarial reasoning as what agent S wants is to make its behaviour as uncertain or unpredictable as possible. In other words,S wants to force the presence of uncertainty in order to confuse the adversary while its payoff is as less affected as possible.

In previous work [2], a model to study the balance between the level of confusion induced and the payoff obtained was proposed. The main conclusion of such study was that one way to produce uncertainty is through decision strategies for S that contain certain amount of randomness. [3] presents a study on automatic design of such strategies. Here we focus on decision strategies that S can use as a means of optimizing his payoffs in situations of repeated conflicting encounters. Essentially we are studying how S can defend against an opponent who is trying to learn their decision rules.

The aim of this contribution is to design and analyze decision strategies for agent S that are not constant along the time, but change at certain time steps in the iterated process. More specifically, we tackle the strategy design as a constrained non-linear optimization problem whose solution gives both the exact moment at which agent S must change and the new strategy he must use.

The contribution is organized as follows. Some basic concepts on adversarial reasoning are outlined in Section II. Section III describes the main characteristics and components of the model used. Section IV deals with the need of randomized strategies. Subsection IV-B explains a static mixed strategy for agent S. Section IV-C introduces the concept of dynamic, time-changing decision strategies for agent S as opposite to the former static strategy. The analytical expression of the expected payoff attained by S when using such a dynamic strategy is given and explained in detail, and the need of an optimization process for determining the best parameters in this expression is motivated. In Section V we describe the computational experiments performed and the results obtained. Finally, Section VI is devoted to discussions and further work.

II. Adversarial Reasoning

As stated before, adversarial decision making is largely about understanding the minds and actions of one's opponent. A typical example is the threat of terrorism and other applications in Defense, but it is possible to envisage less dramatic applications in computer games where the user is the adversary and the computer characters are provided with adversarial reasoning features in order to enhance the quality, hardness and adaptivity of the game. The development of intelligent training systems is also an interesting field.

The threat of terrorism, and in particular the 9/11 event, fueled the investments and interest in the development of computational tools and techniques for adversarial reasoning. However, the field has earlier developments. For example, almost twenty years ago, P. Thagard [4] stated that in adversarial problem solving, one must anticipate, understand and counteract the actions of an opponent. Military strategy, business, and game playing all require an agent to construct a model of an opponent that includes the opponent's model of the agent.

Game theory is perceived as a natural good choice to deal with adversarial reasoning problems. A brief survey of techniques where the combination of game theory with other approaches is highlighted, jointly with probabilistic risk analysis and stochastic games is presented in [1]. Other direct examples that demonstrate in which sense adversarial reasoning (and game theory in particular) can be fully used in real problems are patrolling models for autonomous robots. The aim is to design routes for patrolling trying to minimize the chance that an enemy enters a security border. A lot of research is being done in this area, and several abstract models (with their respective algorithmic solutions) have been proposed so far. Although this topic is not the focus of our particular work, such models bear a clear resemblance to the adversarial reasoning model analyzed in III. In particular, the kind of optimization process applied here was inspired in the one presented in [5]. For further details, please refer to [6] and [7].

III. Adversarial Model

The model we are dealing with was first presented in [2] and it consists of on two agents S and T (the adversary), a set of possible inputs or events $E = \{e_1, e_2, \ldots, e_n\}$ issued by a third agent R, and a set of potential responses or actions $A_i = \{a_1, a_2, \ldots, a_m\}$ associated with every event. We have a payoff or rewards matrix P :

$$P(n \times m) = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ p_{31} & p_{32} & \cdots & p_{3m} \\ \\ p_{n1} & p_{n2} & \cdots & p_{nm} \end{pmatrix}$$

where $p_{ij} \in [0, 1]$ is the reward or profit associated with action j to respond to the event i.

Agent S must decide which action to take given a particular input i_k and with a perfect knowledge of the payoff function P. His aim is to maximize the sum of the profits or rewards given a sequence of inputs. These are issued one at a time and they come from an external environment, represented by agent R. For the experiments, the inputs of the sequence are independent and generated randomly.

Agent T does not know the payoff function P but is watching agent S in order to learn from his actions. His aim is to reduce agent S payoff by guessing which action he will take as a response to each input of the sequence. Algorithm 1 describes the steps of the model, being E the length of the sequence of inputs.

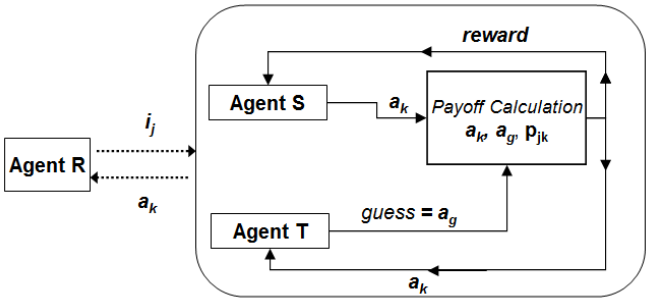

Fig. 1. Graphical representation of the model. Events i_j are issued by agent R while response or actions a_k are taken by agent S.

Algorithm 1 Sequence of steps in the model.

for $j = 1$ to L **do**
  A new input e_j arises.
  Agent T "guesses" an action a_g
  Agent S determines an action a_k
  Calculate payoff for S
  Agent T records the pair e_j, a_k
end for

Given a new input e_j, S and T issue responses a_k and a_g respectively. Agent T keeps records of the actions taken by S using an observation matrix, O, with dimensions $n \times m$. O_{ij} stores the number of times that, in the past, agent S decided to take action a_j when the input was e_i.

The reward calculation for S at stage c is defined as:

$$p' = p_{jk} \times F(a_g, a_k) \tag{1}$$

where F is:

$$F(a, b) = \begin{cases} 0 & \text{if a = b} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

This means that agent S gets no reward when agent T matched his response.

IV. Behaviour of the Agents

In this section, we provide alternatives for modeling the behavior of both agents. For simplicity, we assume that the inputs issued by agent R are equiprobable and that the number of inputs equals the number of actions (i.e, the payoff matrix is square).

A. Strategies for Agent T

Agent T applies a very simple frequency-based decision strategy. Given an event e_i , T uses a strategy called Proportional to the Frequency (PF): the probability of selecting an action j as a prediction to event e_i is proportional to O_{ij} (the observed frequency from agent S) [2].

B. Static Mixed Strategy for Agent S.

Agent S could use a totally deterministic strategy that always select the action with the highest payoff as a response to current stimulus e_i. However, this would be very easy to learn for agent T so he would quickly predict this behavior correctly after a short number. S could also employ a totally random strategy that would select an action in a totally random way. This behaviour would be very hard to learn from observations but, on the other hand, the payoff attained would be low because bad actions (i.e. those with low payoff) may be selected with the same probability than best actions.

The need exists here to get to a good balance between confusion and payoff, as concluded in [2]. But instead of running computational simulations that test new proposals of strategies, our objective now is to calculate the expected payoff of new strategies without running a computational experiment.

In classic game theory, a randomization over the existing actions (responses) is called a *mixed strategy*. A mixed strategy is a set of weights representing a probability distribution over the actions (i.e. the sum of the weights is 1). When a player has to do a movement, he uses this probability distribution to choose his action. In our model, we are interested in the *best randomization*, or in other words, the set of weights that lead to the highest payoff when playing against agent T.

With these weights, it is possible to calculate the so-called expected payoff for a given player, which is the sum of all the possible outcomes of the game weighted by the probability that each outcome eventually arises. In the adversarial model we are dealing with, this means that we can weight each payoff of the payoff matrix P by the probability that agent S eventually gets that payoff. This probability can be computed as the product of the probabilities of several independent events happening simultaneously. Agent S will attain payoff p_{ij} if three conditions hold: (i) Input e_i must arise. We will refer to this probability as $P(I = i)$. (ii) Agent S must select action a_j as a response. This probability is noted α_{ij}. (iii) Finally, S will only get the score p_{ij} if agent T does not successfully predict his response. This probability can be computed as follows.

In case agent S is using a non-variant weight (or probability) α_{ij} during an input sequence of length L to select payoff p_{ij} , then the probability that agent T does not guess his actions if T uses PF strategy is (1-α_{ij}), as explained in the following reasoning. After L_i inputs of a certain kind e_i, since agent S uses α_{ij} , then action a_j will have been selected $L_i \cdot \alpha_{ij}$ times, and this is what agent T sees in O_{ij} . The probability that T selects action a_j as a prediction is then

$$P_{guess} = \frac{O_{ij}}{\sum_{j=1}^{m} O_{ij}} = \frac{L \cdot \alpha_{ij}}{L} = \alpha_{ij} \tag{3}$$

with m being the number of actions available. The probability of not being guessed correctly is then $1-P_{guess} = 1-\alpha ij$.

Taking into account the probabilities of the three conditions described above yields to the following expression of the expected payoff for agent S after a sequence of L inputs when he uses weights α_{ij} to select his actions:

$$EP_{static} = L \cdot \sum_{i=1}^{n} P(I = i) \cdot \sum_{j=1}^{m} \alpha_{ij} \cdot (1 - \alpha_{ij}) \cdot p_{ij} \tag{4}$$

Note that the length of the input sequence L could be inserted into the summation because it does not depend on i. In that case, we could rename the product $L \cdot P(I = i)$ as L_i, the number of times within a sequence of length L that the input was e_i. This notation will be used in the next section. Obviously, $\sum_{i=1}^{n} L_i = L$. If we want to maximize the expected payoff, we have to maximize expression 4 by computing the values of the optimal weights α_{ij} . This can be achieved using numerical optimization methods, such as a gradient descent, subject to two restrictions: (a) $\alpha_{ij} \geq 0$ and (b) $\sum_{j=1}^{m} \alpha_{ij} = 1$. When the events are independent, as we are considering in our model, then the set of optimal weights for each event e_i can be computed separately, because a set of weights does not interact with the rest. This yields to n independent optimization problems (one per input type), each of them having m free variables (one per different action).

The optimization problem for the input type e_i described above can be formalized as follows.

$$\max_{\{\alpha_{ij}\}} \sum_{j=1}^{m} \alpha_{ij} \cdot (1 - \alpha_{ij}) \cdot p_{ij} \tag{5}$$

subject to:

$$\sum_{j=1}^{n} \alpha_{ij} = 1$$
$$\alpha_{ij} \geq 0 \tag{6}$$

The problem above refers only to weights of row i. There are thus n independent optimization problem like (5), one for each row.

C. Dynamic Mixed Strategy for Agent S

In the previous section we described a static strategy for agent S. It was static in the sense that he used all the time the same probability distribution (set of weights) to make a probabilistic decision. We now propose changing these weights along time or, more precisely, at certain moments. A *period* is a series of consecutive inputs for which S will use the same weights to answer. The static mixed strategy described above can be viewed as a single period, because the weights computed by S do not change along time. Now the idea is to define several periods and calculate the optimal mixed strategy for every period. The *length* of a period is the duration of the period, i.e. the number of events during which agent S will use the same mixed strategy.

The next example illustrates this concept. Suppose that we have an input sequence of length L = 1000 stimuli. Then, we can define for instance 4 periods of lengths N^1 = 300, N^2 = 100, N^3 = 200 and N^4 = 400. For a given period, the set of optimal weights can be different from that of other periods. Since the probability of every input to arise is independent

from that of the other inputs, once again we can solve the problem independently for each event. For that reason, we can define a different number of periods of different length for each event, and solve an independent optimization problem for that input. We will call N_i^h the length of the h-th period of event e_i. Fig. 2 shows a different dynamic mixed strategy for each event, with different number of periods and/or different moments in which S switch from a period to the following one at different moments. The length of the whole input sequence is L = 1000. Suppose there exist n = 4 different kinds of inputs in our model. If the random inputs are uniformly distributed, then we could expect that approximately 250 inputs of each event should arise.

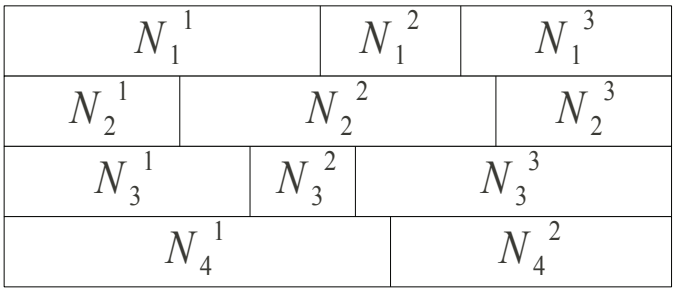

Fig. 2. Example of different periods for each event in a model instance with 4 different events

The expression of the expected payoff for a dynamic strategy can be computed as follows. From now, we will focus only on one single event e_i. Let α_{ij}^h be the set of weights agent S uses to choose an action as a response to an input of type e_i during the h-th period. Then, within a given period, α_{ij}^h represents the probability that S selects action $_j$. The problem now is how to compute the probability of not being guessed, which is the same within a period but different from one period to another. After the first period of length, say, N_i^1, the observation matrix O has the following values in row i (absolute frequencies of the responses given in the past by S to inputs of type e_i):

$$T(n \times m) = \begin{pmatrix} \dots & \dots & \dots & \dots \\ N_i^1 \cdot \alpha_{i1}^1 & \dots & \dots & N_i^1 \cdot \alpha_{im}^1 \\ \dots & \dots & \dots & \dots \end{pmatrix}$$

Clearly, the probability of not being guessed P_{NG} during the first period is (1-α_{ij}^1), according to the same explanation given in section IV-B. This reasoning becomes more complicated when considering row i of the observation matrix at the end of the second period, whose length is N_i^2:

$$T(n \times m) = \begin{pmatrix} \dots & \dots & \dots & \dots \\ N_i^1 \cdot \alpha_{i1}^1 + N_i^2 \cdot \alpha_{i1}^2 & \dots & \dots & N_i^1 \cdot \alpha_{im}^1 + N_i^2 \cdot \alpha_{im}^2 \\ \dots & \dots & \dots & \dots \end{pmatrix}$$

According to the values of the former matrix after 2 periods, the probability *at the end of the second period* that agent T selects action a_j as a prediction is

$$P_{guess} = \frac{O_{ij}}{\sum_{j=1}^m O_{ij}} = \frac{N_i^1 \cdot \alpha_{ij}^1 + N_i^2 \cdot \alpha_{ij}^2}{N_i^1 + N_i^2} \quad (7)$$

so the probability of not being guessed *at the end* of the second period is

$$P_{NG} = 1 - P_{guess} = \frac{N_i^1(1 - \alpha_{ij}^1) + N_i^2 \cdot (1 - \alpha_{ij}^2)}{N_i^1 + N_i^2} \quad (8)$$

What happens in the middle, i.e. *during* the second period? The probability of not being guessed changes at every step because the number of times each response has been observed by T varies along time. This variation can be modeled as follows. At a certain step s of the second period (s is measured from the begining of the period, so $0 \leq s \leq N_i^2$, with N_i^2 being the length of the second period), the probability that T correctly predicts response j to event i is

$$P_{guess} = \frac{O_{ij}}{\sum_{j=1}^m O_{ij}} = \frac{N_i^1 \cdot \alpha_{ij}^1 + s \cdot \alpha_{ij}^2}{N_i^1 + s} \quad (9)$$

and the probability of not being guessed is then

$$P_{NG} = 1 - P_{guess} = \frac{N_i^1(1 - \alpha_{ij}^1) + s \cdot (1 - \alpha_{ij}^2)}{N_i^1 + s} \quad (10)$$

As stated before, notice that this probability changes at every step within a period. Now, it is possible to generalize this reasoning to obtain the probability of not being guessed at step s of the h-th period ($0 \leq s \leq N_i^h$):

$$P_{NG} = 1 - P_{guess} = \frac{\sum_{k=1}^{h-1} N_i^k(1 - \alpha_{ij}^k) + s \cdot (1 - \alpha_{ij}^h)}{\sum_{k=1}^{h-1} N_i^k + s} \quad (11)$$

The expression of the total expected payoff with H_i periods of length N_i^h for every input e_i is a generalization of (4), using (11) as probability of not being guessed:

$$EP_{dynamic} = \sum_{i=1}^{n} P(I = i) \cdot \sum_{h=1}^{H_i} \sum_{s=1}^{N_i^h} \sum_{j=1}^{m} \alpha_{ij}^h \cdot \frac{\sum_{k=1}^{h-1} N_i^k(1 - \alpha_{ij}^k) + s(1 - \alpha_{ij}^h)}{\sum_{k=1}^{h-1} N_i^k + s} p_{ij} \quad (12)$$

Considering L_i as the number of times that event e_i occurred, the next expression should be also verified: $\sum_{k=1}^{H_i} N_i^h = L_i$. This should be an additional constraint in the optimization process that will be carried out to determine the optimal values of all the variables.

The value L_i is unknown and should be estimated in some way[1]. At this stage of the research we assume the following conditions: a) the value L is known, and b) the events are

[1]Static mixed strategies are not affected by this issue because the probability of not being guessed is independent of any external parameter.

generated independently with uniform distribution. So for n events $L_i = L/n \; \forall i \in 1, \ldots, n$.

With this approach, the number of unknown parameters is greater than that of static mixed strategies. The optimal number of periods H_i for a certain event e_i is unknown, and so is the length of each period N_i^h . In addition, instead of computing only m weights for every event (one set of weights per event), we have to compute $m \cdot H_i$ weights (H_i sets of weights per event).

The number of periods H_i we want to introduce is an external parameter which is not part of the optimization process. In order to test the influence of such parameter, we consider H_i varying from 1 to 4 different periods, and compared the results. Recall that the optimal length N_i^h of every period is one of the variables to optimize.

Summarizing, if the events of a sequence are independent, our problem can be broken down in n independent optimization problems, each of them with the following formulation:

$$\max_{\{\alpha_{ij}^h\} \cup \{N_i^h\}} \sum_{h=1}^{H_i} \sum_{s=1}^{N_i^h} \sum_{j=1}^{m} \alpha_{ij}^h \cdot \frac{\sum_{k=1}^{h-1} N_i^k (1 - \alpha_{ij}^k) + s(1 - \alpha_{ij}^h)}{\sum_{k=1}^{h-1} N_i^k + s} p_{ij} \quad (13)$$

subject to:

$$\sum_{j=1}^{m} \alpha_{ij}^h = 1$$

$$\alpha_{ij}^h \geq 0$$

$$\sum_{h=1}^{H_i} N_i^h = L_i = P(I = i) \cdot L \quad (14)$$

A normalization is feasible in expression (13). Instead of using N_i^h, it is possible to optimize the *proportion* of the whole sequence of inputs of a given e_i. This means we can divide every N_i^h by $L_i = \sum_{h=1}^{H_i} N_i^h$. Let γ_i^h be the result of such divisions. It is clear that γ_i^h is always between 0 and 1, and therefore the sum equals 1. As a result, it is possible to execute the optimization process without knowing the number L_i of inputs of type e_i because now the restriction of the sum does not depend on it. However, it is still necessary to know such length when the optimized strategy is to be applied. Otherwise it would be impossible to determine the exact moment in which agent S should switch from using the current set of weights α_{ij}^h to the next set of weights α_{ij}^{h+1}. This yields to the following formulation:

$$\max_{\{\alpha_{ij}^h\} \cup \{\gamma_i^h\}} \sum_{h=1}^{H_i} \gamma_i^h \sum_{j=1}^{m} \alpha_{ij}^h \cdot \frac{\sum_{k=1}^{h} \gamma_i^k (1 - \alpha_{ij}^k)}{\sum_{k=1}^{h} \gamma_i^k} p_{ij} \quad (15)$$

subject to:

$$(a) \sum_{j=1}^{m} \alpha_{ij}^h = 1 \qquad (b) \sum_{h=1}^{H_i} \gamma_i^h = 1$$

$$(c) \alpha_{ij}^h \geq 0 \qquad (d) \gamma_i^h \geq 0 \quad (16)$$

V. Experiments and Results

The experiments we conducted are aimed at answering the following questions

1) Do the results obtained with the analytical expressions match those obtained by empirical simulations?
2) Do dynamic mixed strategies outperform a static mixed strategy in terms of expected payoff?

In order to answer these questions we follow the next steps.

1) Model Configuration: The parameter configuration of the model instance that has been used in the empirical evaluation of strategies was the following:

- Number of different inputs and different actions: n = m = 5
- Length of the input sequences: we checked the expected payoff only for sequences of one event, e_0. The length of every input sequence is $L_0 = 500$.

2) Payoff matrices: 15 different matrices were tested. For each matrix, a set of m payoffs is defined, and every row of the matrix has a permutation of the same set. The payoffs of every matrix are summarized in Table I. The rest of the rows are different permutations of the set displayed in the table.

TABLE I
Set of payoffs associated to each 5x5 payoff matrix. Showing first row of every matrix.

Matrix	First row				
1	1	0,9	0,95	0,8	0,85
2	0,8	0,9	0,6	0,7	1
3	1	0,85	0,7	0,4	0,55
4	1	0,6	0,8	0,4	0,2
5	0,25	0,01	0,5	1	0,75
6	1,1	0,95	0,9	1,05	1
7	1,2	1	1,1	0,9	0,8
8	1,3	1	1,15	0,85	0,7
9	1,2	1,4	1	0,8	0,6
10	1,5	1	0,75	1,25	0,5
11	0,8	0,6	0,4	1,5	1
12	0,8	0,6	0,4	1,75	1
13	0,8	0,6	0,4	2	1
14	0,8	0,6	0,4	2,25	1
15	0,8	0,6	0,4	2,5	1

3) Evaluation of a strategy: every strategy was evaluated by running algorithm 1 100 independent times. The average of the results was taken.

4) Numerical optimization algorithm: an important point is the optimization method employed. Microsoft Excel Solver tool was used to solve every optimization problem. Since every row is a permutation of the same set of values, it is enough to solve the problem for the first row because the weights obtained can be applied (in the correct order) to the others rows. The optimization algorithm implemented by Solver for non-linear optimization problems is called Generalized Reduced Gradient (GRG2); further details can be found in [8].

In order to answer the first question asked above, Fig. 3 shows a comparison of the expected and empirical payoff of an optimal static mixed strategy and an optimal dynamic mixed strategy with 2 and with 4 periods. The plot confirms an almost

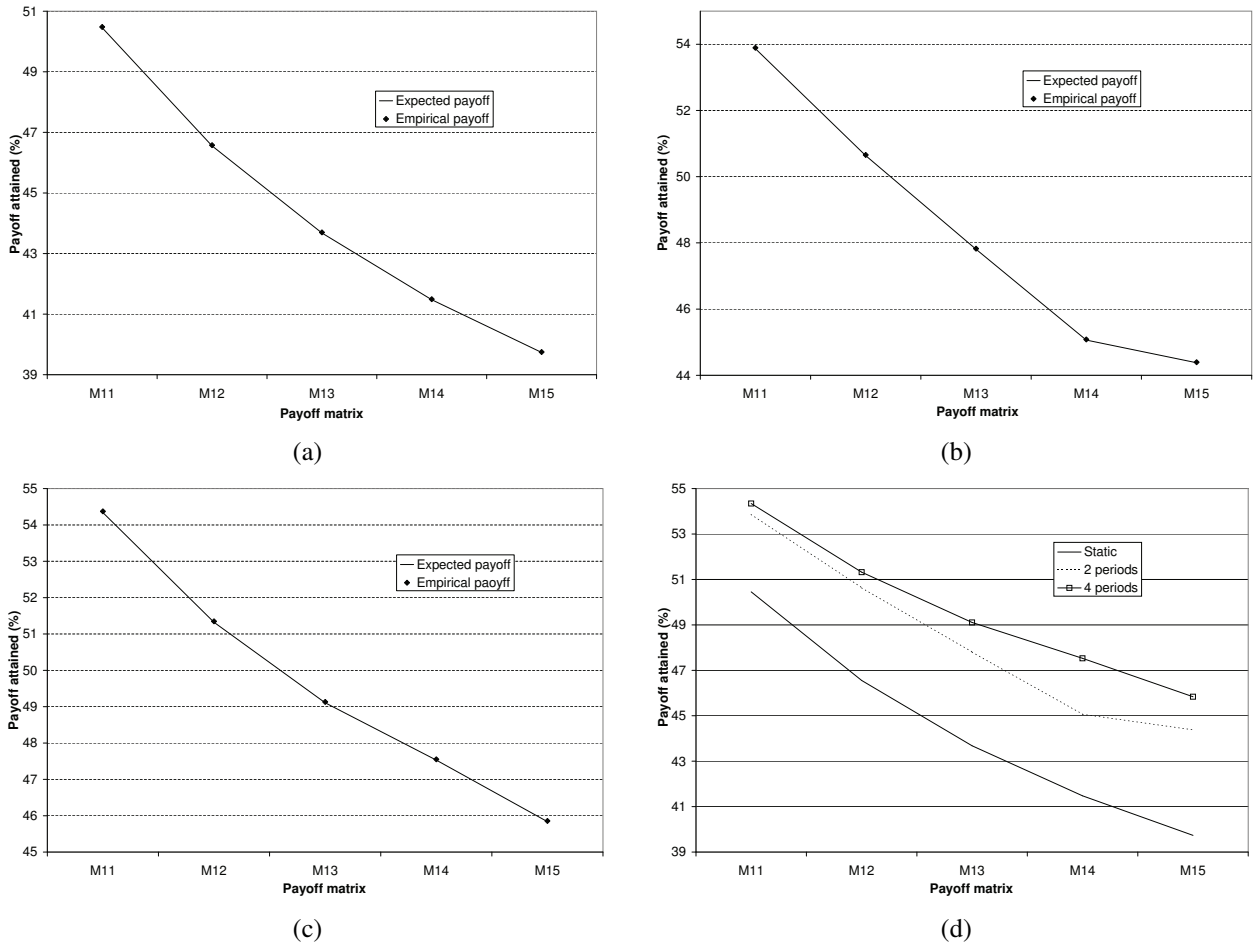

Fig. 3. Expected vs empirical payoff attained by S with static strategies (a) and with dynamic strategies with 2 periods (b) and with 4 periods (c). The gain of using dynamic strategies with 4 periods is shown in (d)

TABLE II
EXPECTED PAYOFF OF STATIC AND DYNAMIC STRATEGIES (WITH 2, 3 AND 4 PERIODS) AFTER 500 EVENTS e_0

Payoff matrix	Maximum	Static	Dyn h=2	Dyn h=3	Dyn h=4
M_1	500	361.26	361.3	363.68	363.61
M_2	500	325.75	333.35	333.43	333.72
M_3	500	295.51	307.95	309.69	308.91
M_4	500	272.08	284.81	286.17	285.70
M_5	500	252.5	269.61	268.534	267.69
M_6	550	401.13	402.92	403.32	403.25
M_7	600	404.56	411.35	411.59	411.76
M_8	650	410.46	422.78	422.59	423.12
M_9	700	419.12	435.92	420.68	436.89
M_{10}	750	430.92	450.30	449.51	450.38
M_{11}	750	378.41	403.99	403.06	407.56
M_{12}	875	407.34	443.05	449.24	449.05
M_{13}	1000	436.79	478.06	490.29	491.05
M_{14}	1125	466.6	506.98	535.36	534.70
M_{15}	1250	496.67	554.73	578	572.96

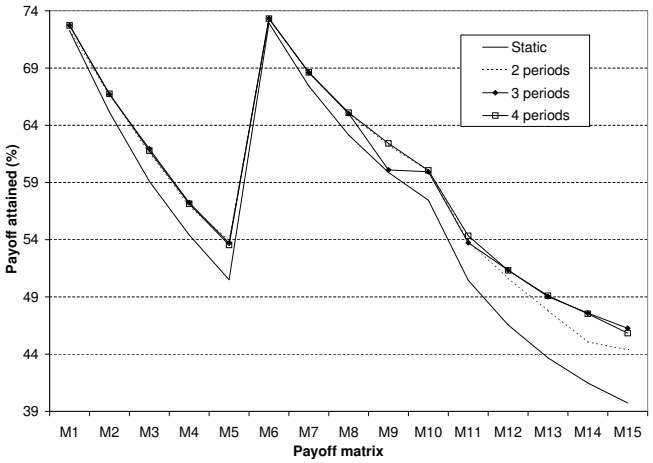

Fig. 4. Predicted payoff for every payoff matrix. Values represent percentages over the maximum

perfect matching between the predicted and the actual payoff attained.

We now analyze the performance of both static and dynamic mixed strategies to answer the second question asked above. Since the expressions have proven to be correct from an empirical point of view, we have just evaluated them for every payoff matrix. The results are shown in Table II and

in Fig. 4. The column of Table II labeled *maximum* stands for the total payoff attainable if agent S always selects the action with the highest payoff and he is never guessed. This would be the ideal situation that only occurs when there is no adversary. The percentages of Fig. 4 were calculated as the actual payoff attained with adversary divided by such *maximum* payoff attainable with no adversary.

Fig. 4 shows a very important result. In all the payoff matrices tested, the three optimal dynamic mixed strategies outperformed the optimal static strategy. In addition, increasing the number of periods was always beneficial in terms of the payoff attained. Recall that these results do not come from a simultation but from a prediction made using the expressions, so they are not influenced by random factors. Notice that the greater gain in performance was achieved in matrices M_{11} to M_{15}, which are those where the highest payoff is much greater than the rest. This is a particularly encouraging result for problems in which it is very important to do the best action as many times as possible.

VI. Conclusion

Static and dynamic mixed strategies for an agent in an adversarial model have been successfully designed using numerical optimization methods. Analytical expressions of the expected payoff for both strategies have been provided and validated also from an empirical point of view. Furthermore, optimal dynamic mixed strategies have shown to outperform optimal static mixed strategies in all the scenarios tested, specially when the difference between the payoff of the best action and the payoff of the rest of actions becomes greater. All these results are encouraging. Further work on this topic may include investigating expressions that do not depend on the prior knowledge of an external parameter about the game that is to be played (in this case, the length of the input sequence), and also more complex time varying strategies that take into account some other conditions of the actual state of the game.

Acknowledgment

This work was supported in part by project TIN2008-01948 from the Spanish Ministry of Science and Innovation and P07-TIC- 02970 from the Andalusian Government.

References

[1] A. Kott and W. M. McEneany, *Adversarial Reasoning: Computational Approaches to Reading the Opponents Mind.* Chapman and Hall/ CRC Boca Raton, 2007.

[2] D. Pelta and R. Yager, "On the conflict between inducing confusion and attaining payoff in adversarial decision making," *Information Sciences*, vol. 179, pp. 33–40, 2009.

[3] P. Villacorta and D. Pelta, "Evolutionary design and statistical assessment of strategies in an adversarial domain," in *Proceedings of the IEEE Conference on Evolutionary Computation (CEC'10)*, 2010, pp. 2250–2256.

[4] P. Thagard, "Adversarial problem solving: Modeling an opponent using explanatory coherence," *Cognitive Science*, vol. 16, no. 1, pp. 123 – 149, 1992.

[5] F. Amigoni, N. Basilico, and N. Gatti, "Finding the optimal strategies for robotic patrolling with adversaries in topologically-represented environments," in *Proceedings of the 26th International Conference on Robotics and Automation (ICRA'09)*, 2009, pp. 819–824.

[6] P. Paruchuri, J. P. Pearce, and S. Kraus, "Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games," in *Proceedings of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS'08)*, 2008, pp. 895–902.

[7] F. Amigoni, N. Gatti, and A. Ippedico, "A game-theoretic approach to determining efficient patrolling strategies for mobile robots," in *Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology (IAT'08)*, 2008, pp. 500–503.

[8] D. Fylstra, L. Lasdon, J. Watson, and A. Waren, "Design and use of the microsoft excel solver," *Interfaces*, vol. 28, no. 5, pp. 29–55, 1998.