Multiagent Learning and Optimality Criteria in Repeated Game Self-play

Andriy Burkov
burkov@damas.ift.ulaval.ca

Brahim Chaib-draa
chaib@damas.ift.ulaval.ca

DAMAS Laboratory
Computer Science Department
Laval University, Canada

Résumé :
Nous présentons une approche d'apprentissage multiagent permettant de satisfaire un critère d'optimalité donné lorsque les jeux répétés se font en "self-play". Notre approche est confrontée aux approches classiques d'apprentissage pour les jeux répétés qui visent généralement à apprendre un équilibre (Nash, Pareto). Une comparaison est donnée d'un point de vue pratique (ou d'ingénieur), c'est-à-dire d'un point de vue d'un concepteur d'un système multiagent dont le but est de maximiser la performance totale du système selon un critère d'optimalité donné. De nombreuses experimentations dans une large variété des jeux répétés démontrent l'efficacité de notre approche.

Mots-clés : Apprentissage multiagent, jeux répétés, jeux en "self-play"

Abstract:
We present a multiagent learning approach to satisfy any given optimality criterion in repeated game self-play. Our approach is opposed to classical learning approaches for repeated games: namely, learning of equilibrium, Pareto-efficient learning, and their variants. The comparison is given from a practical (or engineering) standpoint, i.e., from a point of view of a multiagent system designer whose goal is to maximize the system's overall performance according to a given optimality criterion. Extensive experiments in a wide variety of repeated games demonstrate the efficiency of our approach.

Keywords: Multiagent learning, repeated games, self-play

1 Introduction

Until now, the main body of the state-of-the-art multiagent learning (MAL) research [2] has been focused on finding a learning rule possessing specific properties. For example, when adopted by all agents of a multiagent system (MAS), such a rule could bring to each agent an accumulated reward which is "optimal" in a certain sense. I.e., according to the classical approach, a learning rule is considered to be good if the rewards accumulated by the agents (also called "players") are close to some values satisfying a certain criterion of optimality[1]. Two most widely used optimality criteria in the context of learning in repeated games are: closeness of the value accumulated by each player to the value of a certain (a) Nash equilibrium or (b) Pareto-efficient joint strategy (which need not be an equilibrium).

The scenario where all agents use the same algorithm is called "self-play". Typically, the performance guarantees of a learning rule are given assuming self-play [4, 2, 7, 1]. First of all, this is because it is generally simpler to analyze the properties of a dynamical process induced by a number of identical learning rules. However, another important reason is that the "self-play" multiagent systems (SPMAS) are of a great practical interest.

Indeed, given an algorithm able to converge to a value (or values) close to a given optimality criterion in some SPMAS, an engineer can create a number of identical agents (in the case of software agents, one can just make as many copies of one agent as required) put these agents into a given (usually unknown) environment and let them converge.

However, in an arbitrary repeated game, the values corresponding to different optimality criteria can vary substantially from one criterion to another. So, when the game being played is unknown, it is usually hard to chose the best learning rule. Another problem, when using algorithms satisfying such optimality criteria as (a) or (b) listed above, is that, from a practical point of view, neither of those criteria can be satisfactory for all SPMAS. Let us clarify this claim.

Let suppose that we are an engineer that receives from a client a problem that needs to be solved by a number of identical agents. (We will call this problem "the environment" and this environment is supposed to be unknown in terms of players' rewards for different actions.)

[1] As a matter of fact, the terms "optimal" and "optimality" are not always appropriate in MAS. Indeed, there are often multiple entities (agents) having different interests in a MAS. In the classical game theoretical literature such terms as "Pareto efficiency" or "equilibrium" are used in place of "optimality". Nevertheless, we will use these terms to unify and simplify the presentation.

The agents (if embodied) are provided by the client, but we are free to decide about the algorithms used by the agents to solve the problem. The client expects the good solution to satisfy a certain quantitative criterion based on the values accumulated by the agents. For example, this criterion can require that the solution maximize a given (i.e., provided by the client) algebraic function of players' accumulated rewards. In this case, which of the existing MAL algorithms satisfying their respective optimality criteria will we choose?

One solution would be to run each MAL algorithm on the given problem, observe the results, and pick the best. However, such an approach can be time and ressource expensive, and does not guarantee optimality. A more consistent approach is to construct a new learning algorithm able to solve problems in SPMAS in a way to satisfy functional criteria.

These functional criteria are opposed to such criteria as (a) and (b) above, which we call "relational", meaning that they are defined by taking into account relations between the values accumulated by each individual agent. In this case, the absolute values themselves *are secondary*. For example, a joint-strategy of multiple players is said to be a Nash equilibrium (criterion a) if the expected reward of *each player* is maximized given that *the other players* have their strategies fixed. In a similar manner, a joint-strategy is said to be Pareto-efficient (criterion b) if by changing this strategy so as to increase the expected value of *any subset of players*, there will necessarily be *a player out of this subset* whose value decreases. The same reasoning is applicable to a number of other relational optimality criteria (such as, for example, correlated equilibrium [6]).

In this paper, we propose an approach to multiagent learning in repeated game self-play scenario when the goal is to satisfy a given functional optimality criterion. We show that in such a setting our new learning algorithm, called *Self-play Learner*, is a better choice than a whole family of equilibrium and Pareto-efficient strategy learning algorithms.

2 Formal notions

We focus our attention on repeated games as a model to represent a MAS. For simplicity of exposition, the most of our presentation will be given for two-player case. Extensions to an n-player setting (for an arbitrary $n > 2$) as well as to the multistate problems are discussed in Section 6.

2.1 Matrix games and their solutions

A finite repeated two-player matrix game Γ (henceforth, a repeated game) consists of a set P of two players, p and q, with $(p, q) \in \{(1, 2), (2, 1)\}$, and a set R of two two-dimensional matrices, $R = \{R^p, R^q\}$. Player p has a finite number $M^p \in \mathbb{N}^+$ of actions it can choose from. The game is played iteratively. At iteration $i = 1, 2, \ldots$, each player p chooses an action $a_i^p \leq M^p$ and the vector $\mathbf{a}_i = (a_i^p, a_i^q) \in A$ gives a *joint action*. A is called the joint action space of players. For each player p and to each joint action $\mathbf{a}_i \in A$ there corresponds a real valued number in matrix R^p defining the reward of that player after playing joint action $\mathbf{a}_i$.

To choose an action from M^p at any iteration, each player uses a certain rule. This rule is called player's *strategy*. A player's strategy can be *stationary* or *non-stationary*. Let π_i^p denote the rule by which player p chooses its action at iteration i. Then, p's strategy π^p is called stationary if $\pi_i^p = \pi_0^p$, $\forall i$. This means that p's strategy does not depend on current iteration, or, in other words, that it cannot change with time. Otherwise the strategy is called non-stationary.

A strategy profile $\boldsymbol{\pi}$ is a joint strategy of players, $\boldsymbol{\pi} = (\pi^p, \pi^q)$. To compare strategies and strategy profiles between them (i.e., to say whether one is better than another) it is required to assign a metric to a strategy. We are using the expected limit of the means (ELM) metric. ELM assigns a unique value to an expected sequence of rewards that are obtained by a player when both players follow a given strategy profile $\boldsymbol{\pi}$ during an infinite number of iterations:

$$u^p(\boldsymbol{\pi}) = E_{\boldsymbol{\pi}} \left[\lim_{T \to \infty} \frac{1}{T} \sum_{i=1}^{T} R^p(\boldsymbol{\pi}_i) \right] \quad (1)$$

In the above equation, $u^p(\boldsymbol{\pi})$ is the ELM value of strategy $\boldsymbol{\pi}$. $R^p(\boldsymbol{\pi}_i)$ denotes the expected immediate reward obtained by player p at iteration i if both players follow the strategy $\boldsymbol{\pi}$ at that iteration.

Nash equilibrium is a strategy profile $\hat{\boldsymbol{\pi}} =$

$(\hat{\pi}^p, \hat{\pi}^q)$ such that the following condition holds:

$$u^p(\pi^p, \hat{\pi}^q) \leq u^p(\boldsymbol{\pi}) \text{ and } u^q(\hat{\pi}^p, \pi^q) \leq u^q(\boldsymbol{\pi}), \quad \forall \pi^q \neq \hat{\pi}^q, \pi^p \neq \hat{\pi}^p \tag{2}$$

Let $M(\Gamma)$ denote the set of all strategy profiles of the game Γ. A Pareto-efficient solution of Γ is a strategy profile $\bar{\boldsymbol{\pi}} \in M(\Gamma)$ such that $\forall \boldsymbol{\pi}' \neq \bar{\boldsymbol{\pi}}$, the following condition holds:

$$u^p(\boldsymbol{\pi}') < u^p(\bar{\boldsymbol{\pi}}) \text{ or } u^q(\boldsymbol{\pi}') < u^q(\bar{\boldsymbol{\pi}}) \tag{3}$$

2.2 Optimality criteria

The equations (2–3) define two relational optimality criteria discussed in the previous section. And as we claimed above, there are tasks where a use of relational criteria is not justified from a practical standpoint. In such environments, we would prefer agents to learn (and to use thereafter) the strategies maximizing a certain mathematical function of their utility. This, functional, optimality criterion, depending on the task, can be based on such functions as *max*, *sum*, *product* or any other desirable function of players' individual utilities. If the utility is defined using the ELM metric then the functional optimality criterion $u(\boldsymbol{\pi})$ for a strategy profile $\boldsymbol{\pi}$ can be defined as $u(\boldsymbol{\pi}) = \text{Op}_p(u^p(\boldsymbol{\pi}))$, where $u^p(\boldsymbol{\pi})$ is the utility of player p defined using equation (1). In the latter equation, Op denotes a certain mathematical operator. For a given problem, it needs to be replaced by $\max$, $\sum$, $\times$ or any required function.

We should mention here that the we are not the first to focus on functional optimality criteria. This principle was used by Nash [9] in his axiomatic analysis of bargaining, where he proposed to choose the solution point maximizing the product of individual values of players. From the computer science perspective, Littman and Stone [8] adapted Nash's idea to choose the best solution point in their algorithm computing Nash equilibrium in repeated games. For our part, we extend these ideas to the *learning* in self-play. Another related work is discussed in Section 7.

2.3 Self-play

We *explicitly* focus on the self-play setting; and this is a *controlled* self-play, not an accidental coincidence of learning algorithms of agents. This is what differs our approach from the main body of modern multiagent learning research proposing algorithms whose *behavior is justified for* (or *examined in*) the self-play scenario. Recall that by definition, self-play is a MAS setting in which *all* agents are *identical*. Until now, it has been typically assumed that agents' *algorithms*, or, in other words, rules of strategy update when learning, are identical. Other properties that can also be identical, such as (i) initial knowledge of agents, (ii) their utility metrics and (iii) optimality criteria, have escaped the attention of researchers. In this paper, we aim to fill this gap.

More precisely, in our controlled self-play scenario, which we call SPMAS, we assume that both players, (1) *use the same learning algorithm*, (2) *have internal variables initialized with the values known to both players*, (3) *use the same utility metric* and (4) *optimize the same functional criterion*. We claim that in any *controlled* self-play scenario, Assumptions 2–4 are as well natural as Assumption 1, which is made in many previous multiagent learning papers [4, 2, 7, 6, 1]. In particular, this means that in any SPMAS,

Asmpt. (1) satisfied $\Longleftrightarrow$ Asmpts. (2–4) can be satisfied.

Also, we assume that players can observe each other's actions and their own rewards after a joint action is executed. This is as well a common assumption for many of MAL algorithms [4, 7, 1, 5].

2.4 Information and communication

Two important questions characterizing any MAS are (i) whether the agents know their own reward function and the reward function of the other agent, and (ii) whether communication between agents is available during learning. In this paper, we assume that the answer to both questions is *No*. Indeed, an affirmative answer to the first question makes the learning unnecessary, since the agents can compute an optimal joint strategy using the reward matrices. On the other hand, if there is a communication between agents, the simplest scenario is to explore the reward structure of the game by executing joint actions one by one. Then, using communication, agents are able to share the acquired data. This, again, will make a further learning unnecessary.

3 Extended strategies

To present our new algorithm, we first need to discuss one important implication of using the product criterion: emerging of strategies extended in time, or simply "extended strategies". These non-stationary strategies are known to be able to maximize the product of players' individual utilities to a greater extent than any stationary strategy [8]. As we will demonstrate later, our Self-play Learner algorithm is able to learn extended strategies.

When two players p and q play a joint action $\mathbf{a} = (a^p, a^q)$ their rewards can be visualized as a point $\mathbf{x} = (x^p, x^q) = (R^p(a^p, a^q), R^q(a^p, a^q))$ in a two-dimensional space.[2] Let the set X contain all such points: $X = \{(R^p(a^p, a^q), R^q(a^p, a^q)) : a^p \leq M^p, a^q \leq M^q)\}$. Players can achieve any point in X as their ELM values by playing the corresponding joint action at every iteration. The convex hull of the set X contains all points that can be obtained as a linear combination of a subset of points of X. It is easily observable that the points laying on the boundary of the convex hull are always constructed as a linear combination of only two points of X. In terms of players' strategies, a point $\mathbf{z}$ on the boundary (recall that a point in X is a vector of players' ELM values) can be achieved by the players by playing a joint action, corresponding to a certain point $\mathbf{x}$, a w-fraction of all iterations, and by playing another joint action, corresponding to a point $\mathbf{y}$, the $(1-w)$-fraction of all iterations (where w, $0 \leq w \leq 1$, defines the coefficient of linear combination).

Definition 1. *Given $l \in \mathbb{N}^+$, $0 \leq w \leq 1$, $\mathbf{a} \in A$ and $\mathbf{b} \in A$, an Extended Joint Action (EJA) is a joint strategy in which players play $\mathbf{a}$ during the first $k = \lceil l \cdot w \rceil$ iterations and $\mathbf{b}$ during the following $l - k$ iterations.*

Definition 2. *An Extended Joint Strategy (EJS) is an EJA repeated infinitely often.*

We call l the *length* of an EJA and k its *switch point*. Notice that for each point $\mathbf{z}$ constructed as a combination of two points $\mathbf{x}$ and $\mathbf{y}$ from X, $\exists$ an EJS with certain $\mathbf{a}$, $\mathbf{b}$, l and w.

When the product criterion is used, the boundary of the convex hull is of a particular interest because the point $\mathbf{z}$ maximizing the product criterion is always found on the boundary [9]. For two given points of X, $\mathbf{x} = (x^p, x^q)$ and $\mathbf{y} = (y^p, y^q)$, forming an edge of the boundary, the value of w maximizing the product criterion on this edge can be computed as follows [8],

$$w = \frac{-y^q(x^p - y^p) - y^p(x^q - y^q)}{2(x^q - y^q)(x^p - y^p)} \qquad (4)$$

If $w < 0$ or $w > 1$, the maximum is achieved at respectively $\mathbf{x}$ or $\mathbf{y}$. To find w^* maximizing the product criterion over all points of the convex hull, it is only required go over all pairs of points of X, compute w using Equation (4) and then pick a pair $\mathbf{x}^*$ and $\mathbf{y}^*$ of points (and the corresponding w^*) for which the criterion is maximized.

As one can note, an EJS will achieve the optimal ELM value defined by w^*, $\mathbf{x}^*$ and $\mathbf{y}^*$ only when $l \to \infty$. Let us show that as $l \to \infty$ the error induced by using a finite value of l rapidly decreases.

Proposition 1. *Let R denote the ELM value of an optimal point $\mathbf{z}$ on the boundary of X defined by the values w^*, $\mathbf{x}^*$ and $\mathbf{y}^*$ found as described above. Let l be the length of an EJA defined for two joint actions $\mathbf{a}$ and $\mathbf{b}$ from A corresponding to the points $\mathbf{x}^*$ and $\mathbf{y}^*$ from X. Let $\tilde{R}$ denote the ELM of this EJA. Let $\epsilon = \tilde{R} - R$ define the error of using $l < \infty$ in this EJA. Then $\epsilon \to 0$ as $l \to \infty$.*

Proof. We have $l^2 R = (lwx^p + (l - lw)y^p)(lwx^q + (l - lw)y^q)$ and $l^2\tilde{R} = (\lceil lw \rceil x^p + (l - \lceil lw \rceil)y^p)(\lceil lw \rceil x^q + (l - \lceil lw \rceil)y^q)$. We know that for any natural x, $\lceil x \rceil < x + 1$. Thus, we can write that $l^2\tilde{R} < ((lw + 1)x^p + (l - (lw + 1))y^p)((lw + 1)x^q + (l - (lw + 1))y^q)$. The difference between $l^2\tilde{R}$ and $l^2 R$ is then bounded as follows: $l^2\tilde{R} - l^2 R < l(x^p - y^p)(x^q - y^q)(2w - 1) + 2x^p x^q - x^p y^q - y^p x^q$. Since $l > 1$, the error $\epsilon = \tilde{R} - R$ is bounded as follows: $\epsilon < \frac{(x^p - y^p)(x^q - y^q)(2w-1)}{l} + \frac{2x^p x^q - x^p y^q - y^p x^q}{l^2}$. Therefore, as l tends to ∞, ϵ tends to 0 with a rate inversely proportional to l. □

When l and w^* are known to the players (i.e., defined by the designer of the MAS before to start learning) they are able to construct the EJS maximizing, to the extent of the error induced by using a finite value of l, the product criterion.

[2] In this subsection, we use a simplified notation introduced in [8]. According to it, $\mathbf{x} = (x^p, x^q)$ and $\mathbf{y} = (y^p, y^q)$ denote the vectors of players' rewards for two different joint actions viewed as two points in a two-dimensional space.

4 Self-play Learner

In this section, we present our new algorithm called Self-play Learner (SPL).

4.1 Internal variables

Our algorithm has one internal variable that needs to be initialized with the same value for all agents before the learning is started. This variable, called R_{max}, reflects the maximum utility that an agent can obtain in the game. In many practical tasks, this value can be set by the MAS designer depending on how the utility is defined. For example, for robots cleaning the floor this value can be set based on the maximum possible surface one robot can clean given the initial volume of detergent in its tank. It is assumed that $R_{max} \geq R^p(a^p, a^q)$ for any player p and for all $a^p \leq M^p$ and $a^q \leq M^q$. I.e., R_{max} does not underestimate any of the rewards of players.

During learning, an SPL agent p maintains several other internal variables. The variables $K^p(a^p, a^q)$, $\forall a^p \leq M^p, a^q \leq M^q$, reflect the number of times that a particular joint action (a^p, a^q) has been played. The variables $L^p(a^p, a^q)$, $\forall a^p \leq M^p, a^q \leq M^q$, reflect the number of times that the action a^q has been played by q at the iteration $i+1$ following an iteration i at which players were playing (a^p, a^q).

4.2 Our algorithm

The main steps of our SPL algorithm are:

1. While learning (exploration phase)
   (a) Play a random action,
   (b) Observe the reward R,
   (c) Replay the same action proportionally to R,
   (d) Update counters of the other player's play.
2. While playing (exploitation phase)
   (a) Optimize according to the criterion and the counters,
   (b) Play optimally.

Algorithm 1: Main steps of Self-play Learner.

Exploration phase. During the finite exploration phase (whose length is known by both agents) players explore the reward structure of the game. SPL players are explicitly synchronized (this is an advantage of self-play). During the exploration phase, at each odd iteration i, an SPL player p plays a random action a_i^p and observes its reward R_i^p and the action a_i^q played by the other player. On the next iteration, $i+1$, player p replays the action a_i^p played at the previous iteration with probability $\delta = R_i^p / R_{max}$. Otherwise, with probability $(1-\delta)$ player p plays a random action (different from a_i^p) from M^p.

Exploitation phase. As soon as, during the exploration phase, both players were using the same algorithm, during the exploitation phase player p can make the following assumption about the unknown reward function of player q:

$$\frac{R^q(a^p, a^q)}{R_{max}} \approx \frac{L^p(a^p, a^q)}{K^p(a^p, a^q)}$$

Indeed, since the value R_{max} is the same for (and is known by) both players; and as the players replayed at every iteration $i+1$ their action a_i^p, played at the previous iteration i, according to the proportion $R^p(a_i^p, a_i^q)/R_{max}$, the values of counters $L^p(a^p, a^q)$ and $K^p(a^p, a^q)$ can give to player p a good estimate of the real value of $R^q(a^p, a^q)$. More precisely, player p can compute an estimate of the other player's rewards as follows,

$$\tilde{R}^q(a^p, a^q) = \frac{L^p(a^p, a^q) R_{max}}{K^p(a^p, a^q)} \qquad (5)$$

By so doing, it becomes possible to compute the strategy maximizing any given functional criterion and to execute this strategy thereafter. For example, if the functional criterion is sum, the optimal strategy can be computed by the players as,

$$\pi_i^p = \underset{a^p:(a^p,a^q)\in A \wedge a^q \leq M^q}{\operatorname{argmax}} \left(\tilde{R}^q(a^p, a^q) + R^p(a^p, a^q) \right) \forall i \qquad (6)$$

If the functional criterion is max, the optimal strategy can be computed similarly. Finally, if the functional criterion is $product$, the optimal strategy can be computed as shown in Algorithm 2.

In many games, to construct the optimal strategy to execute during the exploitation phase, both players can use Equation (6) or the procedure of Algorithm 2. The only problem arises when there are several points in X whose ELM values are close to each other according to the functional criterion in question. Let suppose we have

1. For all pairs of points $\mathbf{x}$, $\mathbf{y}$ from the set X, such that, $\mathbf{x} = (R^p(a^p, a^q), \tilde{R}^q(a^p, a^q))$ and $\mathbf{y} = (R^p(b^p, b^q), \tilde{R}^q(b^p, b^q))$ (where $a^p, b^p \leq M^p$ and $a^q, b^q \leq M^q$) compute w using Equation (4) and construct the corresponding EJS using Definitions 1–2.
2. Pick the EJS having the highest ELM value (according to the $product$ functional optimality criterion).

Algorithm 2: Procedure to find the optimal strategy for the $product$ criterion.

a game as follows,

$$R^{1,2} = \begin{pmatrix} 1,2 & 0,0 \\ 0,0 & 2,1 \end{pmatrix}$$

At the end of the exploration phase, players have certain estimates of the other player's reward function computed using Equation (5). Two points, $(1, 2)$ and $(2, 1)$, in X have the same ELM value. However, as the length of the exploration phase is finite, the players have an error in estimates of each other's rewards. Therefore, the strategies computed by the players independently can belong to different joint strategies. For example, player 1 can decide that the optimal strategy is to play the row 1, expecting to see the outcome $(1, 2)$, but player 2 will play the column 2 foreseeing the outcome $(2, 1)$. As the result, they will collect the suboptimal outcome $(0, 0)$.

To settle this coordination problem, SPL players have different roles determined by their player numbers. Roles, as well as player numbers, is a shared information which is given to both players before the learning is started. Each player knows its role and behaves accordingly. Let us call the roles *leader* and *follower*[3][4]. The leader computes its strategy using Equation (6) or the procedure of Algorithm 2. The follower, in order to be synchronized with the leader, computes its strategy as a function of the leader's strategy. In particular, if the functional criterion is sum, the follower computes its strategy as,

$$\pi^p_{i+1} = \underset{a^p:(a^p,a^q_i)\in A}{\operatorname{argmax}} \left(\tilde{R}^q(a^p, a^q_i) + R^p(a^p, a^q_i) \right) \; \forall i \tag{7}$$

If the functional criterion is max, the follower computes its strategy in a similar way. Finally, if the functional criterion is $product$, the follower keeps in memory the most recent EJA played so far. Let $i + 1$ denote the iteration corresponding to the beginning of a new period of length l. At the beginning of iteration $i + 1$, the follower computes the new EJA containing its own sequence of l optimal actions assuming that the leader will follow the strategy played in the previous period. Then, at iterations $i+1, i+2, \ldots, i+l$, the follower executes this sequence. More formally, let H denote the leader's part of the most recent EJA played so far and let H_k, $1 \leq k \leq l$, be the leader's actions in this EJA. To find its optimal sequence of actions, the follower p first finds the switch point k^* as the smallest k such that $H_{k-1} \neq H_k$. Then it sets $a^{q*} = H_{k-1}$ and $b^{q*} = H_k$ and finds the pair (a^{p*}, b^{p*}) that satisfies,

$$(a^{p*}, b^{p*}) = \underset{(a^p,b^p):a^p,b^p\leq M^p}{\operatorname{argmax}} \left(\tilde{R}^q(a^p, a^{q*}) \cdot R^p(b^p, b^{q*}) \right) \tag{8}$$

Then the new EJA is defined by k^* and the joint actions $\mathbf{a}^* = (a^{p*}, a^{q*})$ and $\mathbf{b}^* = (b^{p*}, b^{q*})$.

5 Experimental results

It is only fair to compare a new algorithm with the existing algorithms if it uses the same or relaxed assumptions and is searching for the same kind of solution. In our case, there is no other algorithm capable of learning strategies optimizing functional criteria in MAS (two exceptions and their limitations are discussed in Section 7). On the other hand, there exist a number of MAL algorithms, as those cited above, which, while using different assumptions, converge to the same kinds of relational solutions like Pareto-efficient or Nash equilibrium. So, in our case we will indirectly compare our algorithm with all these algorithms by comparing the ELM value of the solution found by SPL with the corresponding values (according to the same criterion) of different relational solutions. The goal of this comparison is to demonstrate that when the goal of the designer is to satisfy a given functional optimality criterion, SPL is the best choice.

We empirically tested SPL on two different testbeds. The first series of testbeds, called "Random Games M" (or, RGs M, for short), contains randomly generated two-player repeated games with the number of player actions, $M = M^p = M^q$, equal respectively to 2, 3, 5 and 10. In each game from Random Games M, the rewards of players are integer values uni-

[3]Despite such names, the agents are still making their actions simultaneously. Thus we are remaining in a repeated game formalism.

[4]Roles can be assigned at random: the ELM value of a joint strategy does not depend on a particular choice of leader and follower.

formly distributed between 0 and 100, and new values are generated each time a game is started.

The second testbed, called "Conflict Games" (CGs), contains 57 games listed in [3]. These are two-player two-action repeated games whose rewards are integer values between 1 and 4. These games were called "conflict" because there exists no outcome that simultaneously maximizes the ELM value of both players in these games. Conflict Games are especially suitable to make a comparison of solutions computed by SPL for different functional criteria with other possible solutions usually found by other MAL algorithms in self-play (e.g., Nash equilibrium and Pareto-efficient solution).

We did our experiments in the following way. From each testbed, a game was randomly picked and played during $100,000$ iterations. This process (called an *experiment*) was repeated 100 times and then the obtained data were averaged. Table 1 presents the ELM values of the strategies to which SPL players converge in different games. The alternative (relational) solutions are respectively the best (in terms of the corresponding ELM value) pure stationary Pareto-efficient solution (BPE column) and the best and the worst stationary Nash equilibria (BNE and WNE columns respectively). We did not compare the SPL's solution with non-stationary Pareto-efficient solutions because there are no algorithms whose convergence to such kind of solution was proved in a non-special case (one exception is discussed in Section 7). As one can see, in both testbeds the solution found by SPL outperforms all other possible solutions of those games. The advantage of SPL is especially pronounced if the functional optimality criterion is $product$. In this case, SPL often converges to an extended strategy, which is typically more efficient in optimizing this criterion.

The curves of Figures 1 (a–c) reflect the evolution of the ELM value during learning in games from Random Games M. For each learning iteration, the curves present the current ELM value according to one of three functional criteria. (These values were averaged over 100 experiments.) We can observe that for all three functional optimality criteria, the ELM value of SPL becomes close to the optimal one after a reasonably small number of learning iterations.

6 Discussions

So far, we have seen that SPL is efficient in repeated games. Now, let us talk about extensions of SPL to more complex settings, such as n-player, with $n > 2$, repeated games and multistate environments (like those usually modeled as stochastic games [7]). One of possible extensions of SPL to n-player repeated games is quite straightforward. Instead of two roles – leader and follower – there will be n roles, one per player. For example, player 1 assigned with the role 1 will behave as leader. Any other player p assigned with a role $1 < p \leq n$ will behave as follower whose leader is player $p - 1$. Obviously, in this case each player $1 \leq p \leq n$ will maintain the counters L^q and K^q for all other players.

To make an extension of SPL to multistate environments, one could once again take advantage of the self-play setting. Depending on the nature of the environment, agents can use a certain (known to all agents) algorithm to compute a set of strategies for a given environment. For example, for a multi-robot motion coordination problem [7], these strategies can be possible trajectories of a robot. To each of these strategies robots can associate an action of a top level repeated game Γ. Then, a joint-action in the game Γ is a pair of trajectories in the original multistate environment, and the corresponding reward is the cumulative reward of players after simultaneously executing this pair of trajectories.

7 Related Work

We would emphasize two most pertinent works related to our research. In the first one [6] the desired solution of the learning problem is correlated equilibrium. When several equilibria are possible, the author proposes to choose a unique one by using an "objective function", an analog of our functional criterion. However, this implies that agents have two opposite goals: (1) to be selfish (inclination to equilibrium solution implies the agents to be selfish) and (2) to want to sacrifice, by selecting, using the objective function, an equilibrium, which is probably sub-optimal to itself. Generally, if the agents are supposed to want to sacrifice, there is no need in seeking after an equilibrium solution.

In the second work [5], the authors propose an approach for learning of multi-step strategies, analogous to our extended strategies. When the

Table 1: Utility of SPL for different function optimality criteria compared to the utilities of other solutions.

max

	SPL	**BPE**	**BNE**	**WNE**
CGs	**4.00**	3.62	3.44	3.44
RGs 2	**88.71**	82.11	80.02	78.00
RGs 3	**94.68**	88.85	84.19	80.37
RGs 5	**98.18**	94.29	87.83	77.73
RGs 10	**99.46**	96.60	92.68	71.25

sum

	SPL	**BPE**	**BNE**	**WNE**
CGs	**6.41**	6.29	6.02	5.97
RGs 2	**140.73**	126.88	128.10	124.84
RGs 3	**161.14**	151.02	150.34	138.83
RGs 5	**174.05**	162.14	157.98	139.73
RGs 10	**187.15**	182.09	175.93	135.01

product

	SPL	**BPE**	**BNE**	**WNE**
CGs	**10.43**	9.96	9.06	8.94
CGs 2	**5179.85**	4549.06	4432.09	3970.02
CGs 3	**6555.84**	5987.72	5805.01	4720.69
CGs 5	**7715.53**	7050.49	6725.64	5320.81
CGs 10	**8745.05**	8097.79	7806.81	4414.34

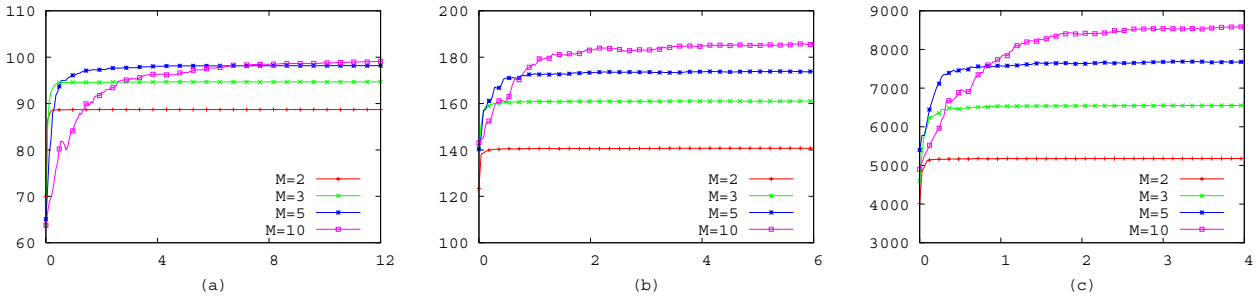

Figure 1: The evolution of ELM value of the learned policy in Random Games M according to different functional criteria: (a) max, (b) sum and (c) $product$. The X axis represents learning iterations ($\times 10^3$); the Y axis represents ELM value.

length l of a multi-step strategy is fixed to 1 (the only value used in their experiments), this yields in a relatively small number of learning iterations. However, by increasing l (to allow more complex joint strategies) the number of joint strategies to explore becomes exponentially large, and only a small number of them is really interesting. In our approach, we find the best extended joint strategies directly, i.e., without enumeration of all pairs of action sequences of length l. Besides, this approach does not permit satisfying a given functional criterion.

8 Conclusions

In this paper, we presented a novel approach to multiagent learning in self-play. We argued that when the learning problem is a known (or controlled) self-play, a good learning algorithm should get additional benefit from this. Then, we presented the notion of functional optimality criterion, as opposed to relational optimality criteria such as Nash equilibrium. We demonstrated that the solution of a problem found by an algorithm seeking to satisfy a relational optimality criterion can be suboptimal if a functional optimality criterion needs to be satisfied. We then showed that in such problems, our algorithm is a better choice than a big class of classical multiagent algorithms for self-play. We also presented the notion of extended strategy and showed how it can be learned. These non-stationary strategies are especially efficient in satisfying the product optimality criterion.

References

[1] B. Banerjee and J. Peng. Performance bounded reinforcement learning in strategic interactions. *Proceedings of AAAI-04*, 2004.

[2] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[3] S.J. Brams. Theory of Moves. *American Scientist*, 81(6):562–570, 1993.

[4] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of AAAI'98*, 1998.

[5] J.W. Crandall and M.A. Goodrich. Learning to compete, compromise, and cooperate in repeated general-sum games. In *Proceedings ICML'05*, 2005.

[6] Amy Greenwald. Correlated-Q learning. In *In AAAI Spring Symposium*, 2003.

[7] J. Hu and M. Wellman. Nash Q-learning for general-sum stochastic games. *Journal of ML Research*, 4:1039–1069, 2003.

[8] M.L. Littman and P. Stone. A polynomial-time Nash equilibrium algorithm for repeated games. *Decision Support Systems*, 39(1):55–66, 2005.

[9] J. Nash. The Bargaining Problem. *Econometrica*, 18(2):155–162, 1950.