Anytime Self-play Learning to Satisfy Functional Optimality Criteria

Andriy Burkov and Brahim Chaib-draa

Laval University, Quebec QC , Canada
burkov@damas.ift.ulaval.ca
http://www.damas.ift.ulaval.ca

Abstract. We present an anytime 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 efficacy of our approach.

1 Introduction

Until now, the main body of the state-of-the-art multiagent learning (MAL) research [1] has been focused on finding a learning rule possessing specific properties. For example, when adopted by all agents of a multiagent system (MAS), such rule could bring to each agent an accumulated reward, "optimal" in a certain sense. I.e., a learning rule were considered to be good if the rewards accumulated by the agents (also called "players") were 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) a Nash equilibrium and (b) a Pareto-efficient joint strategy (which need not be an equilibrium).

The scenario where all agents use the same algorithm is called "self-play". Most of the existing multiagent learning rules assume self-play [2,1,3,4]. An important reason for that is because "self-play" multiagent systems (SPMAS) are of a great practical interest. Indeed, given an algorithm capable of achieving a utility satisfying a given optimality criterion in self-play, 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 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

[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.

F. Rossi and A. Tsoukis (Eds.): ADT 2009, LNAI 5783, pp. 446–457, 2009.

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 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.) 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 algebraic function of player's accumulated rewards. In this case, which of the existing MAL algorithms and their corresponding optimality criteria will we choose?

One solution would be to run each 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. Another approach is to use a learning algorithm capable of solving problems in SPMAS in a way to directly satisfy functional criteria.

These functional criteria are opposed to such criteria as (a) and (b), 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 (for example, correlated equilibrium [5]).

In this paper, we propose an approach to multiagent learning in repeated game self-play when the goal is to satisfy a given functional optimality criterion. We show that in such a setting our learning algorithm, called *Anytime Self-play Learner*, is (i) a better choice than a whole family of equilibrium and Pareto-efficient strategy learning algorithms, and (ii) anytime, i.e., the quality of solution increases gradually with the number of repeated game plays; thus the learning process terminated at an arbitrary moment still results in a reasonably good agents' behavior.

2 Formal Notions

For simplicity of exposition, our presentation will be given for two-player repeated game case. Extensions to an n-player setting (for an arbitrary $n > 2$) as well as to the multistate problems are also possible but omitted due to space limits.

2.1 Matrix Games and Their Solutions

A finite repeated two-player general-sum matrix game Γ (henceforth, a repeated game) consists of a set P of two players, p and q, with $(p, q) \in \{(1, 2), (2, 1)\}$. 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 there is a M^p-by-M^q matrix R^p defining the real-valued reward of that player after playing a joint action $\mathbf{a}_i$.

To choose an action from M^p at any iteration, each player uses its *strategy*. A strategy can be viewed as a function mapping the player p's current internal states into actions. A player's strategy can be *stationary* or *non-stationary*. Let π_i^p denote the rule, following to 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 $\boldsymbol{\pi} = (\pi^p, \pi^q)$ is a joint strategy of players. To compare strategies and strategy profiles between them one can 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 is obtained by a player following a given strategy profile $\boldsymbol{\pi}$ in an infinite sequence of iterations:

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

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:

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

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

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

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) strategies maximizing some 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.

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. The latter property differs our approach from the main body of modern multiagent learning research proposing algorithms whose *behavior is justified for* (or *examined in*) self-play. 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 CSPMAS, 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 [2,1,3,5,4]. In particular, this means that in any SPMAS,

$$\text{Assumption (1) can intentionally be satisfied} \iff \text{Assumptions (2–4) can intentionally be satisfied.}$$

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 [2,3,4,6]. To stay as much general as possible, in our approach we assume that the reward observed by an agent p, $\forall (p,q) \in \{(1,2),(2,1)\}$, at iteration i after playing a joint action $\mathbf{a}_i$, is not necessarily deterministic. We only suppose that $\forall \mathbf{a} \in A$, the reward $R_i^p(\mathbf{a})$ observed after playing a joint action $\mathbf{a}$ is an instantiation of a random variable with certain mean and variance that do not change with time.

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. Accordingly, if the agents can communicate, 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 [7]. As we will demonstrate later, our Anytime Self-play Learner algorithm is able to learn and execute 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 $\mathbf{z}$ obtained 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 this criterion is always found on the boundary [8]. 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 [7],

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

If $w < 0$ or $w > 1$, the maximum is achieved at respectively $\mathbf{x}$ or $\mathbf{y}$. To find w^* maximizing the product criterion, 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 for which w 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*

[2] In this subsection, we use a simplified notation introduced by Littman [7]. 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 points in a two-dimensional space.

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 $l^2R = (lwx^p + (l-lw)y^p)(lwx^q + (l-lw)y^q)$ 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^2R is then bounded as follows: $l^2\tilde{R} - l^2R < l(x^p - y^p)(x^q - y^q)(2w-1) + 2x^px^q - x^py^q - y^px^q$. Since $l > 1$, the error $\epsilon = \tilde{R} - R$ is bounded as follows: $\epsilon < \frac{(x^p-y^p)(x^q-y^q)(2w-1)}{l} + \frac{2x^px^q - x^py^q - y^px^q}{l^2}$.

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.

4 Anytime Self-play Learner

In this section, we present our new algorithm called Anytime Self-play Learner (ASPL). Its main steps are given in Algorithm 1.

Algorithm 1. Main steps of Anytime Self-play Learner

1. While exploring
   (a) Play an exploration 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 exploiting
   (a) Optimize according to the criterion and counters,
   (b) Play optimally.

4.1 Internal Variables

Our algorithm has one internal variable that needs to be initialized to all agents with the same value 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 designer depending on how the utility is defined. E.g., for floor cleaning robots 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 ASPL agent p also maintains several other 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 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 Exploring

To exhibit a good joint behavior, players obviously need to explore the game they play. More specifically, an agent not only needs to learn its own rewards for joint actions, which, as we noticed, can be an instantiation of an unknown random variable. Also, in order to optimize the functional criterion, it needs to have "an idea" of the reward function of the other agent.

For this purpose, the algorithm proceeds as follows. Both ASPL players are explicitly synchronized (this is an advantage of self-play). At each odd iteration i, an ASPL player p randomly uniformly plays an action a_i^p, and observes its own reward R_i^p and the action a_i^q played by the other player. Then it updates its estimate $\tilde{R}^p(\mathbf{a}_i)$ of the reward $R^p(\mathbf{a}_i)$ for the joint action $\mathbf{a}_i = (a_i^p, a_i^q)$ as follows:

$$\tilde{R}^p(\mathbf{a}_i) = \tilde{R}^p(\mathbf{a}_i) + \frac{1}{k_{\mathbf{a}_i} + 1}\left(R_i^p - \tilde{R}^p(\mathbf{a}_i)\right) \tag{5}$$

where $k_{\mathbf{a}_i}$ represents the number of times the joint action $\mathbf{a}_i$ has been played so far. Finally, player p increments its counter $K^p(\mathbf{a}_i)$.

At the next iteration, $i + 1$, player p replays the action a_i^p with probability $\delta = R_i^p / R_{max}$. Otherwise, with probability $(1 - \delta)$ player p plays a random action different from a_i^p and picked uniformly from the remaining actions. Player p then observes the action played by player q, updates again its reward estimate for the played joint action using Equation (5), and, finally, updates its counter $L^p(\cdot)$ accordingly.

4.3 Exploiting

As soon as when exploring both players follow the same procedure, therefore, at any moment of time, player p can assume for any joint action $(a^p, a^q) \in A$ that $\frac{L^p(a^p,a^q)}{K^p(a^p,a^q)}$ is an unbiased estimator of the unknown reward function of player q. More precisely, it can believe that,

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

Indeed, since the value R_{max} is the same and is known to both players, and as the players replayed their action a_i^p, played on 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 rewards as follows,

$$\tilde{R}^q(a^p, a^q) = \frac{L^p(a^p, a^q)R_{max}}{K^p(a^p, a^q)} \tag{6}$$

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,

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

Similarly, if the functional criterion is max, the optimal strategy can be computed as,

$$\pi_i^p = \underset{a^p:(a^p,a^q)\in A}{\operatorname{argmax}} \left(\max(\tilde{R}^q(a^p,a^q), \tilde{R}^p(a^p,a^q))\right) \ \forall i \tag{8}$$

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

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

1. For all pairs of points $\mathbf{x}$, $\mathbf{y}$ from the set X, such that, $\mathbf{x} = (\tilde{R}^p(a^p,a^q), \tilde{R}^q(a^p,a^q))$ and $\mathbf{y} = (\tilde{R}^p(b^p,b^q), \tilde{R}^q(b^p,b^q))$ (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).

5 Explore, Exploit and Coordinate

As one could remark, ASPL can be viewed as an anytime algorithm. This means that every two iterations the agents improve their estimates of the reward functions. If the learning (exploring) is stopped after a certain iteration, the players are able to choose the best strategy as yet and to play on it. However, in a general case two important issues need to be resolved in order to assure a good joint performance. These can be formulated as two questions: (1) when to stop exploring and start exploiting (known as the exploration-exploitation dilemma [9]) and (2) how to choose a coordinated joint strategy (i.e., the coordination problem [2]).

While the exploration-exploitation dilemma is a well-known problem in machine learning, several words need to be said about the coordination problem in MAS. Let suppose we have a game as follows,

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

After a certain number of exploration iterations, players have certain estimates of their reward functions. Two points, $(1,2)$ and $(2,1)$, in X have the same ELM value. However, as the number of exploration iterations is always finite, the players have an error in estimates of their own and 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 $(0,0)$.

In practice, the designer of CSPMAS can sufficiently know the environment in order to be able to apply the "explore-then-exploit" principle [10]. More precisely, this means to separate the learning process into two phases (exploration and exploitation) and to fix the length of the exploration phase before the learning starts. For example, the room to clean by two robots can be the same, but the positions of chairs and tables can change. In

this case, the designer can know that the agents need to continuously explore during the first T time steps, and then they can synchronously switch to exploit. The coordination problem can also be easily solved then. For instance, one agent can be assigned to play the role of *leader*, i.e., to choose the best joint strategy. The second agent, in turn, will act as *follower*, i.e., it will observe the strategy followed by the leader at the previous iteration and adapt to it. Such roles can be assigned at random since the ELM value of a joint strategy does not depend on a particular choice of leader and follower.

5.1 Mixed Exploring and Exploiting

On the other hand, if the designer is not able to fix in advance the length of the exploration phase, he can prefer to mix exploration and exploitation in a certain way, and let agents "decide" online. One technique to do this involves using a GLIE (for "greedy in the limit with infinite exploration") learning strategy. An example of such a strategy is Boltzmann exploration strategy:

$$\pi_i^p(a^p) = \frac{e^{\beta_i Q_i(a^p)}}{\sum_{b^p \leq M^p} e^{\beta_i Q_i(a^p)}}$$

where $\pi_i^p(a^p)$ denotes the probability with which player p chooses to play the action a^p at each odd iteration i, β_i is an exploration factor slowly increasing with time (Singh et al. [11] show how it can be defined in order to assure convergence in the limit). Q_i can be viewed as a certain current preference of agent p over its actions. In our experiments, to solve the coordination problem, we compute $Q_i(a^p)$ at each odd iteration using a heuristic called Combined Optimistic Boltzmann (COB) proposed by Claus and Boutilier [2]. It consists of assigning higher values to the "promising" strategies weighed by the likelihood to be simultaneously chosen by the other player. As the exploration decreases (with the increasing factor β) this heuristics permits the agents to effectively coordinate on the same joint strategy and to play on it most of the time. Notice that there exist more efficient methods (in terms of exploration time and cost) to mix exploration and exploitation, such as a Bayesian approach [12].

6 Experimental Results

It is only fair to compare a new algorithm with the existing ones 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 ASPL 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, ASPL is the best choice.

We empirically tested ASPL 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 uniformly 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 by Brams [13]. These are two-player two-action repeated games whose rewards are integer values between 1 and 4. These games were called "conflict" because contain no outcome that simultaneously maximizes the ELM value of both players. Conflict Games are especially suitable to make a comparison of solutions computed by ASPL 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 conducted 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 ASPL players converge in different games. The alternative values are respectively (APE column) the average utility (in terms of the corresponding ELM value) of all pure stationary Pareto-efficient solutions and (ANE column) the average utility of all stationary Nash equilibria found by the Lemke-Howson algorithm. We did not compare the ASPL'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 ASPL outperforms all other solutions of those games. The advantage of ASPL is especially pronounced if the functional optimality criterion is $product$. In this case, ASPL often converges to an extended strategy, which is typically more effective in optimizing this criterion.

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

max				sum				$product$			
	ASPL	**APE**	**ANE**		**ASPL**	**APE**	**ANE**		**ASPL**	**APE**	**ANE**
CGs	**4.00**	3.62	3.44	CGs	**6.41**	6.29	6.02	CGs	**10.36**	8.29	9.09
RGs 2	**88.71**	82.11	80.02	RGs 2	**140.73**	126.88	128.10	CGs 2	**5179.85**	4512.07	4537.66
RGs 3	**94.68**	88.85	84.19	RGs 3	**161.14**	151.02	150.34	CGs 3	**6555.84**	4190.74	4672.07
RGs 5	**98.18**	94.29	87.83	RGs 5	**174.05**	162.14	157.98	CGs 5	**7715.53**	5432.45	5652.87
RGs 10	**99.46**	96.60	92.68	RGs 10	**187.15**	182.09	175.93	CGs 10	**8745.05**	6466.64	6305.67

The curves of Figures 1 (a and b) 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[3] according to the sum and $product$ functional criteria (the curves for max look similar and were omitted due to the space limits). We can observe that for each functional optimality criterion, the ELM value of ASPL becomes close to the optimal one after a reasonably small number of learning iterations.

[3] These values were averaged over 100 experiments.

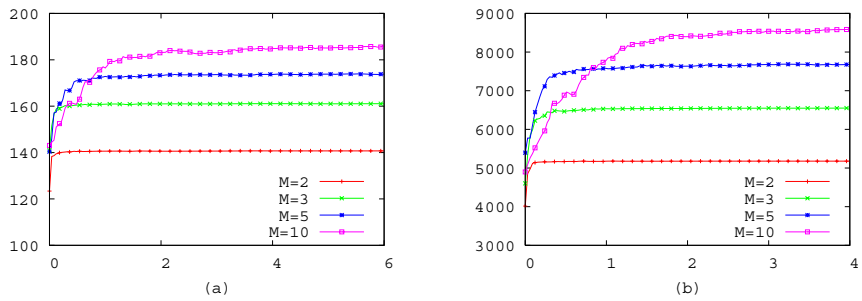

Fig. 1. The evolution of ELM value of the learned policy in Random Games M according to different functional criteria: (a) sum and (b) $product$. The X axis represents learning iterations ($\times 10^3$); the Y axis represents ELM value.

Both "explore-then-exploit" and "mixed exploring-exploiting" strategies of exploration/coordination performed well in our experiments. The only concern about the latter strategy is that GLIE principle guarantees the convergence to an optimal behavior only in the limit. Further, since COB is a heuristic, in a relatively small number of runs agents could coordinate on a suboptimal strategy. However, we observed that even in such rare cases, agents still chose a joint strategy "close" to an optimal one and never the worst one.

7 Related Work

Here, we would emphasize two related works. In the first one, by Greenwald [5], 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, Crandall and Goodrich [6] propose an approach to the learning of multi-step strategies, analogous to our extended strategies. When the length l of an extended joint action 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 extended joint actions and thereby obtain other values inside the convex hull) the number of joint strategies to explore becomes exponentially large, but only a (very) 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, the approach of Crandall and Goodrich does not permit satisfying a given functional criterion.

8 Conclusions

In this paper, we presented a novel approach to learning in self-play. We argued that when the learning problem is a controlled self-play, a good learning algorithm should get additional benefit from this. E.g., the agents can have internal variables initialized with the same values, use the same utility metric and optimize the same function. We then presented the notion of functional optimality criterion, as opposed to relational optimality criteria such as Nash equilibrium. We pointed out that the solution of a problem found by an algorithm seeking to satisfy a relational criterion can be suboptimal if a functional optimality criterion needs to be satisfied. We demonstrated that in such problems, our algorithm is a better choice than the classical algorithms for self-play.

References

1. Bowling, M., Veloso, M.: Multiagent learning using a variable learning rate. Artificial Intelligence 136(2), 215–250 (2002)
2. Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. In: Proceedings of AAAI 1998 (1998)
3. Hu, J., Wellman, M.: Nash Q-learning for general-sum stochastic games. Journal of ML Research 4, 1039–1069 (2003)
4. Banerjee, B., Peng, J.: Performance bounded reinforcement learning in strategic interactions. In: Proceedings of AAAI 2004 (2004)
5. Greenwald, A.: Correlated-Q learning. In: AAAI Spring Symposium (2003)
6. Crandall, J., Goodrich, M.: Learning to compete, compromise, and cooperate in repeated general-sum games. In: Proceedings ICML 2005 (2005)
7. Littman, M., Stone, P.: A polynomial-time Nash equilibrium algorithm for repeated games. Decision Support Systems 39(1), 55–66 (2005)
8. Nash, J.: The Bargaining Problem. Econometrica 18(2), 155–162 (1950)
9. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
10. de Farias, D., Megiddo, N., Cambridge, M., San Jose, C.: Exploration-Exploitation Tradeoffs for Experts Algorithms in Reactive Environments. In: Advances in Neural Information Processing Systems 17: Proceedings of The 2004 Conference. MIT Press, Cambridge (2005)
11. Singh, S., Jaakkola, T., Littman, M., Szepesvári, C.: Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. Machine Learning 38(3), 287–308 (2000)
12. Chalkiadakis, G., Boutilier, C.: Coordination in multiagent reinforcement learning: A bayesian approach. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003), Melbourne, Australia (2003)
13. Brams, S.: Theory of Moves. American Scientist 81(6), 562–570 (1993)