Journal of Machine Learning Research 9 (2008) 2635-2675 Submitted 7/08; Published 12/08

Multi-Agent Reinforcement Learning in Common Interest and Fixed Sum Stochastic Games: An Experimental Study*

Avraham Bab BAB@CS.BGU.AC.IL
Ronen I. Brafman BRAFMAN@CS.BGU.AC.IL
Department of Computer Science
Ben-Gurion University
Beer-Sheva 84105, Israel

Editor: Michael Littman

Abstract

Multi Agent Reinforcement Learning (MARL) has received continually growing attention in the past decade. Many algorithms that vary in their approaches to the different subtasks of MARL have been developed. However, the theoretical convergence results for these algorithms do not give a clue as to their practical performance nor supply insights to the dynamics of the learning process itself. This work is a comprehensive empirical study conducted on *MGS*, a simulation system developed for this purpose. It surveys the important algorithms in the field, demonstrates the strengths and weaknesses of the different approaches to MARL through application of FriendQ, OAL, WoLF, FoeQ, Rmax, and other algorithms to a variety of fully cooperative and fully competitive domains in self and heterogeneous play, and supplies an informal analysis of the resulting learning processes. The results can aid in the design of new learning algorithms, in matching existing algorithms to specific tasks, and may guide further research and formal analysis of the learning processes.

Keywords: reinforcement learning, multi-agent reinforcement learning, stochastic games

1. Introduction

Multi-Agent Reinforcement Learning (MARL) deals with the problem of learning to behave well through trial and error interaction within a multi-agent dynamics environment when the environmental dynamic and the algorithms employed by the other agents are initially unknown. Potential applications of MARL range from load balancing in networks (Schaerf et al., 1995) and e-commerce (Sridharan and Tesauro, 2000) to planetary exploration by mobile robot teams (Zheng et al., 2006).

MARL adopts the game theory model of a Stochastic (a.k.a. Markov) Game (SG) to model the multi-agent-environment interaction. The non-cooperative[1] game theoretic solution concept for SGs is the Nash Equilibrium (NE). A NE is a behavioral profile, namely a set of decision rules, or policies, for all agents, such that no agent can benefit from unilaterally changing its behavior. However, SGs may have multiple NEs with different values, none of which is necessarily strictly optimal (i.e., preferable by all agents to all other NEs). Thus, in the general case, it is not clear which behavior should be considered "optimal," even when the environmental dynamics and the other players' set of possible strategies are known. For this reason, development of MARL algorithms

*. A preliminary version of this paper that covered some of the results on common-interest games appeared in Bab and Brafman (2004).

1. In this context, the meaning of 'non-cooperative' is that agents are selfish and do not collaborate or communicate with other agents, except through the game.

has concentrated on algorithms for classes of SGs in which there is a unique NE, or in which all NEs have the same value. In such cases, it is possible to measure the performance of learning algorithms against a well defined target.[2] In particular, most MARL algorithms are shown to converge to such NEs in self play in either Common Interest SGs (CISGs) or Fixed Sum SGs (FSSGs), which we describe next.

CISGs model environments in which the agents share common interests and have no conflicting interests. In such environments, defining an optimal joint behavior for all agents is straightforward—it is the joint behavior that maximizes the common interests. However, since the agents are independent, they face the task of coordinating such joint behavior in the case in which there are several optimal options. FSSGs, on the other hand, model environments in which two agents have fully conflicting interests. In FSSGs, there is a well defined *minimax* solution (Filar and Vrieze, 1997).

Several different MARL algorithms have been proved to converge in the limit to optimal behavior in CISGs (Littman, 2001; Wang and Sandholm, 2002) and in FSSGs (Littman, 1994). One has been shown to converge to ε-optimal behavior in polynomial time in both CISGs and FSSGs (Brafman and Tennenholtz, 2002, 2003). Since MARL is, by its nature, an online task, determining the abilities of the algorithms in practical domains is important. However, existing theoretical results tell us very little about the practical efficacy of the algorithms;[3] to this end a comprehensive empirical comparison is necessary. Experimental results that have been published in the literature on CISGs (Claus and Boutilier, 1997; Wang and Sandholm, 2002; Chalkiadakis and Boutilier, 2003) and on FSSGs (Littman, 1994; Uther and Veloso, 2003; Bowling and Veloso, 2002), do not meet this demand. They do not examine representative samples of algorithms and/or use small and simple test models and/or do not examine online learning. Furthermore, the different experimental setups used in different publications do not enable cross comparisons of the algorithms they examine.

This work provides a comprehensive empirical study of MARL algorithms in CISGs and FSSGs. It offers a decomposition of the MARL task into subtasks. It then compares three algorithms for learning in CISGs: FriendQ (Littman, 2001), OAL (Wang and Sandholm, 2002), and Rmax (Brafman and Tennenholtz, 2002); and three algorithms for learning in FSSGs: FoeQ (Littman, 1994, 2001), WoLF (Bowling and Veloso, 2002) and Rmax (Brafman and Tennenholtz, 2002). These algorithms were selected because they represent a variety of approaches to the offered subtasks, while providing certain convergence guarantees. We experimented with diverse variants of these algorithms on several non-trivial test environments which we designed to demonstrate the efficacy of the different approaches in each of the subtasks. To concentrate attention on the basic learning task, full state observability and perfect monitoring (that is, the ability to observe the actions of other agents) are assumed. The results allow us to rank the performance of the algorithms according to properties of the environment and possible performance measures.

The experiments for this work have been conducted using MGS, a Markov Game Simulation system developed for this purpose. MGS is implemented in the Java programming language and supplies interfaces and abstract classes for the simple creation of players and grid worlds and convenient logging. We believe that MGS can be of good service to both MARL algorithm designers and users. MGS is free, open source software available at `http://www.cs.bgu.ac.il/~mal`.

2. Much recent work is concerned with the question of how to define and evaluate the performance of learning algorithms in more general games. See, for example, Vohraa and Wellman (2007) which is devoted to this issue.
3. Vidal and Durfee (2003) take a step towards theoretical analysis of the learning dynamics. They offer theoretical tools to analyzing and predicting behavior of multi-agent systems that are represented by simpler models than SGs. Powers and Shoham (2005) offer experimental results on iterative games, which are a much simpler model than SGs.

The paper is organized as follows. Necessary background is given in Section 2. Sections 3 and 4 describe the particular problems and algorithms for CISGs and FSSGs, respectively, and present experimental results and analysis. Section 5 describes MGS and Section 6 concludes the paper.

2. Multi-Agent Reinforcement Learning and Stochastic Games

Multi-Agent Reinforcement Learning (MARL) is an extension of RL (Sutton and Barto, 1998; Kaelbling et al., 1996) to multi-agent environments. It deals with the problems associated with the learning of optimal behavior from the point of view of an agent acting in a multi-agent environment. At the outset, the environmental dynamics and the algorithms employed by the other players are unknown to the given agent. The environment is modeled by a finite set of states and the agents-environment interaction is discretized into time steps. At each time step, the players simultaneously choose actions, available from individual sets of actions. Depending stochastically on the joint action, the environment transitions into its next state and each player is rewarded. The present work assumes full state observability and perfect monitoring, namely, the agent observes the actions taken and rewards received by the other players. It also assumes that the agents have no additional means of communication. The multi-agent-environment interaction is modeled by a Stochastic (a.k.a Markov) Game (SG).

Definition 2.1 (Stochastic Game) *An SG* $G := \{\alpha, A, S, T, R\}$ *consists of:*

- $\alpha = \{1, ..., n\}$ *- a set of players. We will typically use n to denote the number of players.*
- $A = A_1 \times A_2 \times ... \times A_n$ *– a set of joint actions.* A_i *is a set of private actions available to player i.*
- S *- a set of states.*
- $T : S \times A \times S \to [0,1]$ *- a transition function.* $T(s,a,s') = Pr(s' \mid s,a)$ *is the probability that the system transitions to state* s' *when joint action a is taken at state s (*$\sum_{s'} T(s,a,s') = 1$*).*
- $R : S \times A \times S \to \mathbb{R}^n$ *- a payoff function.* $[R(s,a,s')]_i$ *is i's reward upon transition from state s to state* s' *under joint action a.*

The behavior of player i in an SG is described by a *policy*. A policy is a mapping $\pi_i : \mathscr{H} \to \mathscr{PD}(A_i)$ where $\mathscr{H} := \{(s_0, a_1, s_1, a_2, ..., s_j) \mid j \geq 0\}$ is the set of possible histories of the process and $\mathscr{PD}(A_i)$ is a probability distribution over A_i. A policy that depends only on the current state of the process, that is, $\pi_i : S \to \mathscr{PD}(A_i)$ is called *stationary*. A deterministic policy, that is a mapping, $\pi_i : \mathscr{H} \to A_i$ is called *pure*, whereas a stochastic policy is called *mixed*. A tuple of policies $\pi = (\pi_1, \ldots, \pi_n)$ for n players of a SG is called a *policy profile*. The objective of an agent in a SG is to maximize some function of its accumulated payoffs, referred to as the agent's return. In this study, the infinite horizon discounted return (IHDR) is considered. The expected IHDR for player i, resulting from policy profile π, is defined by the sum $\sum_{t=0}^{\infty} \gamma^t E^{\pi}(r_t^i)$ where r_t^i is player i's payoff at time t and $\gamma \in [0,1)$ is a discount factor. Consequently, a state-policy value function, V is defined by $V_i(s,\pi) = \sum_{t=0}^{\infty} \gamma^t E^{\pi}(r_t^i \mid s_0 = s)$.

We note that different algorithms optimize different objectives. Yet, typically, the same underlying ideas can be used to formulate different variants of the same basic algorithm that aim to

maximize different natural objectives. While we use formulations that aim to maximize IHDR, the games we experiment on are such that any good policy will reach an absorbing state (following which the agents are placed in their initial states) quickly. In this setting, given a reasonably high discount factor, γ, IHDR maximizing behavior will be identical to behavior maximizing average reward. Consequently, we will sometimes find it more natural to report performance measures such as average reward per step.

For single agent domains, where $n = 1$, there is always an optimal pure stationary policy that maximizes $V(s,\pi)$ for all $s \in S$ (Filar and Vrieze, 1997). The single-agent state-policy value function for the optimal policy, referred to as the state-value function, is the unique fixed point of the Bellman optimality equations

$$V^*(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s') \right), \forall s \in S \, .$$

An optimal policy may be specified by $\pi^*(s) = arg\max_{a \in A}(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s'))$ (Puterman, 1994). Many single agent Reinforcement Learning (RL) methods interleave approximation of the value function with derivation of a learning policy from the current approximation.

In MARL, maximizing the IHDR cannot be done by simply maximizing over (private) policies since the return depends also on the other players' policies which, in turn, may depend on the agent's actions. Hence, to maximize the IHDR, the agent must adopt a policy that is a *best response* to the other players' policies. Formally, π_i is a best response to $\pi^{-i} = (\pi_1, ..., \pi_{i-1}, \pi_{i+1}, ..., \pi_n)$ if $V_i(s, \pi_1, ..., \pi_i, ..., \pi_n) \geq V_i(s, \pi_1, ..., \pi_i', ..., \pi_n)$ for all π_i' and $s \in S$. A best response function is defined by $BR(\pi^{-i}) = \left\{ \pi_i \mid \pi_i \textit{ is a best response to } \pi^{-i} \right\}$. In general, $\bigcap_{\pi^{-i}} BR(\pi^{-i}) = \emptyset$, namely, there is no policy that is a best response to all of the possible behaviors of the other players.

Whereas the goal of single-agent reinforcement learning is clear—maximizes some aggregate of your reward stream, the picture is more complex in multi-agent settings. Here, one's performance depends on what the other agents do, and strategic considerations come to the fore. For instance, the well-known notion of Nash Equilibria does not, in general, provide a clear target for learning algorithms, as many such equilibria may exist in a game, none of which dominates the others. Although some recent work has attempted to clarify this issue (Brafman and Tennenholtz, 2004; Shoham et al., 2007), there is still no clear agreement on the goal of MARL. However, there are two special classes of SGs in which there is a clear target for learning: Common-interest SGs, and Fixed-sum SGs. These are two extreme cases of SGs where players are either fully cooperative or fully opposed. Much work in the area of MARL has concentrated on these classes of SGs, and algorithms with good theoretical guarantees exist for each of them. In this paper, we analyze a number of algorithms for such games.

3. Learning in CISGs

In CISGs, the payoffs are identical for all agents. That is, for any given choice of s, a and s' and any pair i, j of agents, we have that $[R(s,a,s')]_i = [R(s,a,s')]_j$. Therefore, all agents have identical interests and we may speak of optimal joint policies, namely, policy profiles that maximize the common IHDR for the team of agents. Such profiles are also NEs because no agent can gain by deviating from them. CISGs pose all the standard challenges of single-agent RL, in particular the need to balance exploration and exploitation and to propagate new experience. In addition, they challenge the agents to coordinate behavior since to obtain maximum value may require that agents

select a particular joint action. On the other hand, CISGs do not require that agents confront the more difficult task of optimizing behavior against an adversary.

For efficient learning in CISGs, agents are required to coordinate on two levels: (i) select whether to explore or exploit in unison; and (ii) coordinate the exploration and exploitation moves. This requirement stems from the dependence of the team's next state on the actions of *all* its members. Hence, it is impossible for the team to exploit unless all agents exploit together, and using the same choice of exploitation strategy. Exploration, too, can be less effective when only some agents explore.

Furthermore, even when the model is known, multiple NEs yielding maximal payoffs to the agents are likely to exist, and the agents still face the task of reaching consensus on which specific NE to play.

This section describes and compares three algorithms for learning in CISGs: OAL (Wang and Sandholm, 2002), FriendQ (Littman, 2001), and Rmax (Brafman and Tennenholtz, 2002). They were selected because each embodies a different approach to learning, while guaranteeing convergence to optimal behavior in CISGs. Diverse variants of these algorithms are examined with the aim of gaining better understanding of their performance with respect to their approach to exploration-exploitation, information propagation, and coordination tasks.[4] These variants and the tasks on which they were tested are described in the following subsections.

3.1 FriendQ

FriendQ (Littman, 2001) extends single agent Q-learning into CISGs. After taking a joint action $a = (a_1, ..., a_n)$ in state s at time t and reaching state s' with reward r_{cur}, each agent updates its Q-value estimates for $\langle s, a \rangle$ as follows:

$$Q_t(s,a) \leftarrow (1-\alpha_t)Q_{t-1}(s,a) + \alpha_t \left(r_{cur} + \gamma \max_{a' \in A} Q(s', a') \right).$$

As in single agent Q-learning, given that $\sum_{t=0}^{\infty} \alpha_t = \infty$, $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$ and that every joint action is performed infinitely often in every state, the Q-values are guaranteed to converge asymptotically to Q^* (Littman, 2001). Convergence to optimal behavior is achieved using *Greedy in the Limit with Infinite Exploration Learning Policies* (GLIELP) (Sutton and Barto, 1998).

There are two types of GLIELPs, *directed* and *undirected*. Directed GLIELPs reason about the uncertainty of the current belief about action values (Kaelbling, 1993; Dearden et al., 1998, 1999; Chalkiadakis and Boutilier, 2003). However, the computational complexity of the underlying statistical methods makes directed exploration impractical for simulations of the size conducted in this study.[5] Two popular undirected exploration methods are ε-greedy action selection and Boltzman distributed action selection. There is no established technique for applying Boltzman exploration to FriendQ, so in our experiments it is executed with ε-greedy exploration only. ε-greedy exploration is applied to SGs in the following way: each agent randomly picks an exploratory private action with probability ε, and with probability $1-\varepsilon$ takes its part of an optimal (greedy) joint action with

4. By this we mean the ability of the algorithm to propagate information observed in one state to other states. For example, Q-learning does not propagate information beyond the current state, unless techniques such as eligibility traces are used.

5. It can be argued that many realistic applications impose severe constraints on the length of trajectories. In this case, directed exploration techniques and techniques such as transfer learning appear to be essential for success. Conducting a study of algorithms for such contexts would seem to be of great interest.

respect to the current Q-value (Claus and Boutilier, 1997). ε is asymptotically decreased to zero over time.

Since full state observability, perfect monitoring, and identical initial Q-values to all agents are assumed, all agents maintain identical Q-values throughout the process, and consequently the same classification of greedy actions. But, two problems arise: (i) Because randomization is used to select exploration *or* exploitation, the agents cannot coordinate their choice of when and what to explore. (ii) In the case of multiple optimal policies, that is, several joint actions with maximal Q-values in a certain state, the agents must agree on one such action. The original FriendQ algorithm has no explicit mechanism for handling these issues.

This work compares some enhanced versions of FriendQ: First, Uncoordinated FriendQ (UFQ), the simple version described above, is tested. Next, the effect of adding coordination of greedy joint actions by using techniques introduced by Brafman and Tennenholtz (2003) is examined. Basically, a shared order over joint actions is used for selecting among equivalent NEs. If such an order is not built into the agents, it is established during a preliminary phase using an existing technique (Brafman and Tennenholtz, 2003). This version is referred to as Coordinated FriendQ (CFQ). Then, coordination of exploration and exploratory actions is added in Deterministic FriendQ (DFQ). In DFQ, the agents explore and exploit in unison, always exploring the least tried joint action. An exploratory action is taken each $\lfloor 1/\varepsilon \rfloor' th$ move. Finally, we add Eligibility Traces (Sutton and Barto, 1998) to DFQ (ETDFQ).[6] Eligibility Traces propagate new experience to update Q-values of previously visited states and not only the most recently visited state.

3.2 OAL

OAL combines classic *model-based* reinforcement learning with a new *fictitious play* algorithm for action and equilibrium selection named BAP (Biased Adaptive Play) (Wang and Sandholm, 2002). BAP is an action-selection method for a class of repeated games that contains common interest games. Here, BAP is described in the context of common interest repeated games. Let m and k be integers such that $1 \leq k \leq m$. Each agent maintains a memory of the past m joint actions. At the first m steps of the repeated game, each player randomly chooses its actions. Starting from step $m+1$, each agent randomly samples k out of the m most recent joint actions. Let SP_i be the set of k joint actions drawn by agent i at some time step. If (i) there is a joint action a' that is estimated to be ε-optimal, such that for all $a \in SP_i$, $a^{-i} \subset a'$ (where $a^{-i} \subset a'$ denotes the fact that the individual actions of all agents other than agent i are identical in a and a'), and (ii) there is at least one optimal joint action $a \in SP_i$, then agent i chooses its part of the most recent optimal joint action in SP_i. If the above two conditions are not met, then agent i chooses an action a_i that maximizes its expected payoff under the assumption that the other players' sampled history reflects their future behavior. This type of action selection is known as *fictitious play* (Brown, 1951).

$$EP(a_i) = \sum_{a^{-i} \in SP_i} R(a_i \cup a^{-i}) \frac{N(a^{-i}, SP_i)}{k}$$

where $N(a^{-i}, SP_i)$ is the number of occurrences of a^{-i} in SP_i. Given that there is no sub-optimal NE and $m \geq k(n+2)$, BAP is guaranteed to converge to a NE. It was shown that, for every game that satisfies these conditions, there is some positive probability p and some positive integer T such that

6. A variant of Eligibility Traces called Replacing Traces was used (Singh and Sutton, 1996). In Replacing Traces, the eligibility traces are bounded by 1.

for any history of plays, with probability at least p, BAP converges to a consensus in T steps. That is, all players agree in the same joint-action which is a NE.

After observing a transition from state s on action a, OAL updates Q-values according to the learning rule

$$Q_{t+1}(s,a) = R_t(s,a) + \gamma \sum_{s'} T_t(s,a,s') \max_{a'} Q_t(s',a')$$

where R_t, the approximated mean reward, and T_t, the approximated transition probability are estimated using the statistics gathered up to time t. At each step, OAL constructs a Virtual Game (VG) for the current state-game (the matrix game defined by the current state's Q-values) and plays according to it. The VG has common payoff 1 for any optimal joint action and payoff 0 for any other action. In our implementation we use the VG in conjunction with ε-greedy as well as Boltzman action selection. Boltzman action selection is implemented as follows: At each step, an action is sampled according to the Boltzman distribution induced by the Expected Payoffs in the current VG

$$\frac{e^{EP(s,a)/\tau}}{\sum_b e^{EP(s,b)/\tau}} .$$

If a sub-optimal action is sampled, it is explored by the agent, otherwise BAP is executed on the VG to select an exploitation action.

We examine OAL also with an addition of Prioritized Sweeping (PS) (Moore and Atkeson, 1993) to the underlying Q-learning algorithm (PSOAL). PS is a heuristic method for optimizing finite propagation of TD-errors in the model. PS attempts to order propagation according to the size of the change to the Q-values, for example, states that are liable to have a greater update should be updated first. For comparison, a combination of the model-based Q-learning algorithm used by OAL with the action and equilibrium-selection technique used by CFQ is also examined. This combination is referred to as ModelQ (MQ).

3.3 Rmax

Rmax (Brafman and Tennenholtz, 2002) is a model-based algorithm designed to handle learning in MDPs and in fixed-sum stochastic games. However, because Rmax does not make random decisions (e.g., random exploration), its MDP version can also be used to tackle MARL in CISGs. Brafman and Tennenholtz (2003) view a CISG as an MDP controlled by a distributed team of agents and show how such a team can coordinate its behavior given a deterministic algorithm such as Rmax. In a preliminary phase of the game, a protocol is used to establish common knowledge of the individual action sets, of orders over these sets, and of an order over the agents. At each point in time, all agents have an identical model of the environment and know what joint action needs to be executed next (when a number of actions are optimal with respect to the current state, the agents use the shared order over joint actions to select among these actions). Thus, each agent plays its part of this action. It is shown that even weaker coordination devices can be used, and that these ideas can be employed even under imperfect monitoring.

Rmax maintains a model of the environment, initialized in a particular optimistic manner. It always behaves optimally with respect to its current model, while updating this model (and hence its behavior) when new observations are made. The model M' used by Rmax consists of $n+1$ states $S' = \{s_0, ..., s_n\}$ where $s_1, ..., s_n$ correspond to the states of the real model M, and s_0 is a fictitious state.[7] The transition probabilities in M' are initialized to $T_{M'}(s,a,s_0) = 1 \ \forall \langle s,a \rangle \in S' \times A$. The

7. The model may be constructed online as states are discovered.

reward function is initialized to $R_{M'}(s,a) = R_{max}\ \forall \langle s,a\rangle \in S' \times A$, where R_{max} is an upper bound on $\max_{s\in S, a\in A} R(s,a)$. Each state/joint-action pair in M' is classified either as *known* or as *unknown*. Initially, all entries are unknown.

Rmax computes an optimal policy with respect to M' and follows this policy until some entry becomes known. It keeps the following records: (i) number of times each action was taken at each state and the resulting state; (ii) the actual rewards, $r_{ac}(s,a)$, received at each entry. An entry (s,a) becomes known after it has been sampled $K1$ times, such that with high probability $T_M(s,a,s') - \rho \leq PE(s,a,s'|K1) \leq T_M(s,a,s') + \rho$ where T_M is the transition function in M, $PE(s,a,\cdot\ |K1)$ is the empirical transition probability according to the $K1$ samples, and ρ is the accuracy required from M'. When an entry (s,a) becomes known, the following updates are made: $T_{M'}(s,a,\cdot) \leftarrow PE(s,a,\cdot\ |K1)$ and $R_{M'}(s,a) \leftarrow r_{ac}(s,a)$. Then, a new deterministic optimal policy with respect to the updated model is computed and followed. Rmax converges to an ε-optimal policy in polynomial number of steps.

The worst-case bounds on $K1$ (Brafman and Tennenholtz, 2002) assume maximal entropy on the transition probabilities, that is, $T_M(s,a,s') = 1/|S|$ for all s,a,s'. These bounds, although polynomial, are impractical. In the experiments, these bounds are violated, which enables us to eliminate knowledge of the state space size. Furthermore, R_{max} is not assumed to be known. Instead, it is initialized to some positive value and updated online to be twice the highest reward encountered so far.

3.4 Discussion of Algorithms

Returning to the FriendQ algorithm, the efficiency of GLIELPs depends on the topology and dynamic of the environment. If the probability to explore falls low before "profitable" parts of the environment are sufficiently sampled, the increasing bias to exploit may keep the agents in suboptimal states. As a result, GLIELPs can exhibit significant differences depending on the particular schedule of exploration. In model free algorithms, and FriendQ, in particular, this phenomenon is intensified by the decreasing learning rate that makes learning from the same experience slower over time. GLIELPs also suffer from their inability to completely stop exploration at some point. Thus, even when greedy behavior is optimal, the agent is unable to attain optimal return.

The exploration method of Rmax is less susceptible to the structure of the environment. As long as Rmax cannot achieve actual return ε-close to optimal, it will have a strong bias for exploration since unknown entries seem very attractive. This strategy is profitable when the model can be learned in a short time. However, the theoretical worst-case bounds for convergence in Rmax are impractical. In practice, much lower values of $K1$ suffice. Bayesian exploration (Dearden et al., 1999; Chalkiadakis and Boutilier, 2003) and locality considerations might help to obtain better adaptive bounds, but these approaches are not pursued here.

GLIELPs make learning "slower" as the agents get "older". To accelerate learning, an algorithm can try to use new experience in a more exhaustive manner, using it to improve behavior in previously visited states. Eligibility traces are used to propagate information in FriendQ. In model-based algorithms, an exhaustive computation per new experience is too expensive (in CPU time). Thus, OAL is tested with Prioritized Sweeping and Rmax makes one exhaustive computation each time a new entry becomes known (and does no further computation).

Exploration in FriendQ and OAL algorithms is not coordinated. Each of the agents independently chooses an exploratory action with some diminishing probability. Thus, joint actions that

have no element (private action) of some optimal joint action have a lower chance of being explored. Hence, some popular techniques for decreasing exploration in the single agent case lead to finite exploration in the multi agent case. For example, taking $\varepsilon = 1/time$ for ε-greedy policies will make the chance of exploring such joint actions $1/time^n$, where *n* is the number of agents.

Equilibrium selection in Rmax and CFQ comes with no cost. In OAL, it is essentially a random protocol for achieving consensus. This protocol may take long to reach consensus with respect to the current Q-values, but provides for another exploration mechanism at early stages, when Q-values are frequently updated.

All three algorithms have parameters that need to be preset. Parameter tuning is task specific and based more on intuition and trial and error than on theoretical results. FriendQ has a range of parameters for decaying the learning rate, the exploration probability and the eligibility traces, which also pose inter-parameter dependencies. For decreasing the learning rate parameter, we used the results presented in Even-Dar and Mansour (2003). OAL takes parameters for history sample size and for exploration. In this respect, Rmax is friendlier. It has a single and very intuitive parameter—number of visits to declare an entry *known*. When the value of this parameter is high, a very accurate model is learned and behavior will be, eventually, very close to optimal. But this comes at the cost of possibly unnecessary exploration and delayed exploitation.

Table 1 summarizes the differences between the three algorithms according to the features mentioned above.

property	**UFQ**	**CFQ**	**DFQ**	**OAL**	**Rmax**
Exploration	ε-greedy with exponential and polynomial decay of ε			Boltzman & ε-greedy (polynomial decay)	Greedy w.r.t optimistic model
Coordination	None	Common order; non-deterministic	Common order; deterministic	Random protocol; non-deterministic	Common order; deterministic
Greedy Action Selection	Maximize common return			Fictitious play/consensus	Maximize common return
Information Propagation	Single sweep per step & ET			Single sweep per step & PS	Limited exhaustive computations
Parameter Tuning	Many parameters, task sensitive, not intuitive				One parameter, not sensitive, intuitive

Table 1: Major differences between the experimented algorithms.

3.5 Experimental Results & Analysis

This section describes experiments with the FriendQ, OAL, and Rmax algorithms on three CISGs. The games were designed to evaluate the effects of exploration, coordination, and information-propagation methods on performance in different environments. All games are grid-based. The grid cells are referred to by *(row, column)* coordinates indexed from $(0,0)$ at the top left corner of the grid. In all games, the available actions for each agent are *up*, *down*, *left*, *right*, and *stand*. The games were played in both deterministic and stochastic modes. In deterministic mode, the action always succeeds. In stochastic mode, each action, excluding *stand*, succeeds with probability 0.6. With probability 0.4, uniformly at random, the agent is moved to the each of the other adjacent cells or left in place. Action *stand* succeeds with probability 1. If the direction of motion is towards a wall, the player remains in place. Similarly, two players cannot occupy the same position. Therefore, if

two agents attempt to move into the same cell, they both fail and remain in their current place. Note that in stochastic mode, these rules apply to the actual (stochastic) outcome of the action.

Additionally, for each game, we examined the results of learning by heterogeneous agents, that is, agents using different learning algorithms. Finally, to test how well each algorithm scales up with the number of players, we introduced a fourth game in which the state and action spaces do not grow too fast with the number of players. With this game, we were able to play games with up to 5 players.

Adjusting the parameters of the different algorithms was done by a process of trial and error. The algorithms were repeatedly executed in an experimental setup, varying their parameters between executions until some optimum was reached. The parameters that achieved the best performance were then used throughout. Each set of experimental conditions, other than those related to Rmax, was subjected to 100 repeated trials. For Rmax, 20 trials were carried out using $K1 = 50$, 40 with $K1 = 100$ and 40 with $K1 = 200$.[8] The discount factor was 0.98 in all trials. Unless mentioned otherwise, the presented results are averages over all trials.

The following parameter settings were tested:

FriendQ

Exploration: ε-greedy with (i) $\varepsilon_t \leftarrow 1/count_t^{0.5000001}$ where $count_t$ is the number of exploratory steps taken by time t. (ii)$\varepsilon_t \leftarrow 0.99998^{count_t}$. (Unless specified otherwise, (i) is used.)[9]

Learning rate: $\alpha_{s,a} \leftarrow 1/n(s,a)^{0.5000001}$ where $n(s,a)$ is the number of times action a was taken in state s.

Q-value were initialized to 0.

OAL

Exploration: For ε-greedy, $\varepsilon_t \leftarrow 1/count_t^{0.5000001}$, as in FriendQ. For Boltzman exploration, the temperature parameter was decreased by $\tau \leftarrow 100/count^{0.7}$.

History: Random history sample size $k = 5$. History memory size $m = 20$ (m must satisfy $m \geq k \times (n+2)$. OAL with ε-greedy exploration is referred to as ε-OAL, and OAL with Boltzman exploration is referred to as B-OAL.

Q-value were initialized to 0.

Rmax

Sampling: values of 50, 100, 200 and 300 for K1 (visits to mark an entry known) were tested.

Accuracy of Policy Iteration: Offline policy iteration was halted when the difference between two successive approximations was less than 0.001.

8. Because Rmax is a deterministic algorithm, fewer samples were required.
9. Exponential decay of ε violates the infinite exploration condition for convergence.

3.5.1 GAME 1

This game, introduced in Hu and Wellman (1998), was devised to emphasize the effects of equilibrium-selection methods. It has a single goal state (the only reward-yielding state) and several optimal ways of reaching it. The game is depicted in Figure 1. $S(X)$ and $G(X)$ are the respective initial and goal positions of agent X. In the goal state G, both agents are in their goal positions and their reward is 48. Upon reaching the goal, the agents are reset to their initial position. The underlying SG has 71 states. The optimal behavior in deterministic mode reaches G in four steps and yields an average reward per step (a.r.p.s.) of 12.[10] There are 11 different optimal equilibria. In stochastic mode, the optimal policies yield an a.r.p.s. of $\sim$5.285. Algorithms were executed for 10^7 rounds on both settings.

S(A)		G(B)
S(B)		G(A)

Figure 1: Game 1 - initial and goal states.

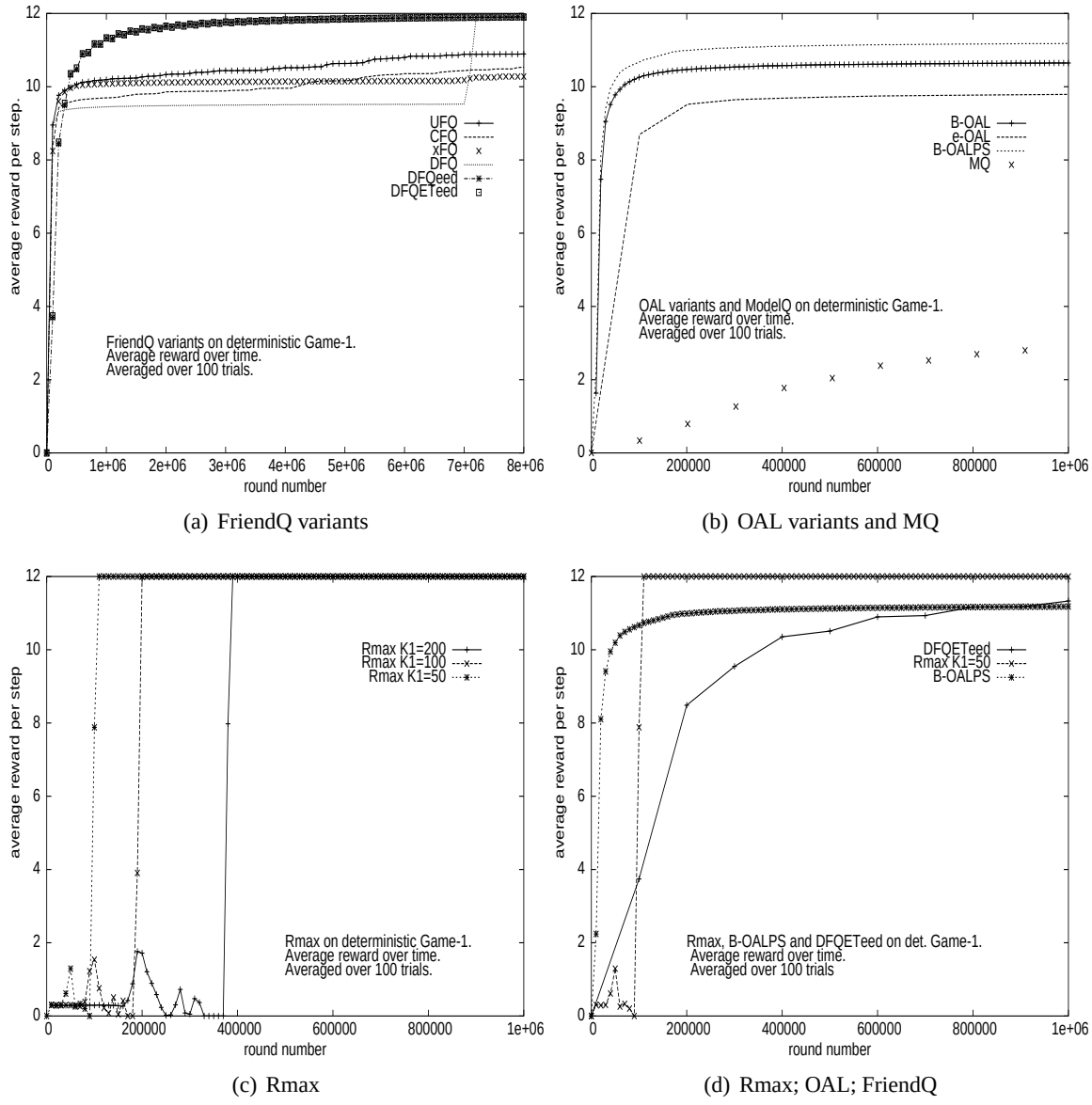
Deterministic Mode

Table 2 reports the number of trials (of 100 in total) in which each algorithm learned a policy, with four levels of final performance based on the number of steps required to reach the goal. For this deterministic domain, we find this measure, which is directly correlated with the more standard a.r.p.s. measure, to be more informative. Here xFQ is a variant of FriendQ in which the agents explore in unison but do not coordinate exploratory actions. The suffix "εed" denotes exponential decay of ε. In the present context, the agents' learning of an optimal policy means that their greedy choice of actions is optimal. That is, with any residual exploration deactivated. Figure 2 presents the a.r.p.s. obtained by the agents over time.

steps to goal	UFQ	CFQ	xFQ	DFQ	DFQεed	ε-OAL	B-OAL	B-OALPS	MQ	Rmax
4	62	49	47	100	100	26	49	41/60	1	100
5	38	49	46			62	51	19/60	49	
6+		2	7			12			29	
∞									21	

Table 2: Game 1 – classification of final performance of learned policies for 100 trials of each algorithm

10. As we noted earlier, our implementation is based on the widely used discounted reward model. But in our goal oriented domains, optimal and near-optimal strategies require a relatively small number of steps to reach the goal. Thus, we chose to report the performance of the learned policies using more intuitive measures such as average reward per step and average steps to reach the goal.



(a) FriendQ variants

(b) OAL variants and MQ

(c) Rmax

(d) Rmax; OAL; FriendQ

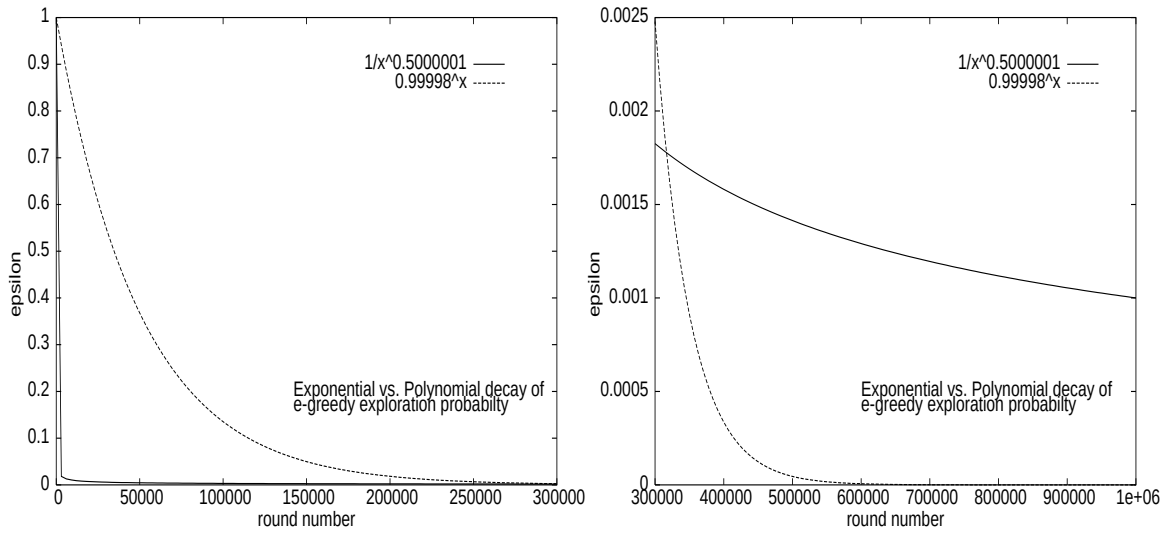
Figure 2: Game 1 – average reward per step under deterministic mode. (a) presents first 8×10^6 rounds. (b), (c) and (d) present first 10^6 rounds.

As can be inferred from the table, in this problem optimal behavior is such that the agents reach the goal in four steps. FriendQ converges quickly to second-best behavior (Figure 2a). From that point on, the average learning curves of UFQ, CFQ and xFQ increase stepwise rather than continuously (although this is a bit difficult to see in the figure). This behavior results from a sudden switch of the FriendQ agents from sub-optimal to optimal behavior once the relative order of the Q-values of different agents changes.

ε-OAL does not present a similar trend. In the trials in which OAL converged to second-best behavior in the first 2.5×10^5 rounds, it failed to find an optimal policy even after 10^7 rounds (Figure 2b). In DFQ, since exploration is deterministic, this switch is always at the same time, specifically after 7×10^6 rounds(Fig. 2a).

Surprisingly, UFQ fares better than CFQ (Table 2, Fig. 2a), in spite of its less sophisticated coordination strategy. At an early learning stage, dis-coordination leads to exploration. Later on, the estimated Q-values of optimal actions are rarely equal, and thus, coordinating exploitation does not pose a problem (at the examined time interval). Exponential decay of ε supplies more exploration at an early period than polynomial decay (Fig. 3) leading to faster convergence of DFQεed (Fig. 2a).

Eligibility traces did not contribute much in this example. The parameters of eligibility traces were hard to tune and very sensitive to change in other parameters or environment dynamics.



(a) Rounds 1 to 300,000

(b) Rounds 300,000 to 1,000,000

Figure 3: Exponential vs. polynomial decay of ε-greedy exploration probability

OAL agents converge relatively quickly to optimal or second-best behavior, and from that time onwards stick to their behavior (Fig. 2b). Whether, in the latter case, they fail to structure the Q-values properly, or the fictitious play prevents the agents from changing their behavior after the Q-values are ordered properly, is not clear from the data. B-OAL converges faster and more often to optimal than ε-OAL (Fig. 2b, Table 2). This behavior seems to stem from the effect of the decay methods we used. The Boltzman method yields more exploration than the ε-greedy method in the early period of learning. Later on, ε-greedy maintains a low exploration probability that decays very slowly while Boltzman exploration drops faster to zero. Thus, even when ε-OAL learns optimal behavior, it keeps achieving only near-optimal average-reward.

As expected B-OALPS improves on the performance of B-OAL (Fig. 2b, Table 2) because of its more rapid propagation of learned information.

The performance of ModelQ is inferior to that of OAL (Fig. 2b, Table 2), presumably because ModelQ does not explore as much as OAL: At early stages of learning, fictitious play provides OAL with other means of exploration. When the agents make many stochastic action choices in

early stages of learning, fictitious play amplifies the random behavior. However, at later stages of learning, deviation from constant action choice is rare and will probably not affect fictitious play. In this setting, ModelQ shows slower convergence than the model-free FriendQ.

The learning graph of Rmax can be precisely divided into two periods, an initial learning period in which Rmax attains very low return due to exploration, followed by a period of exploitation in which Rmax attains optimal return (Fig. 2c). The length of the initial period depends linearly on $K1$.[11] Figure 2d compares the best performing variant of each algorithm.

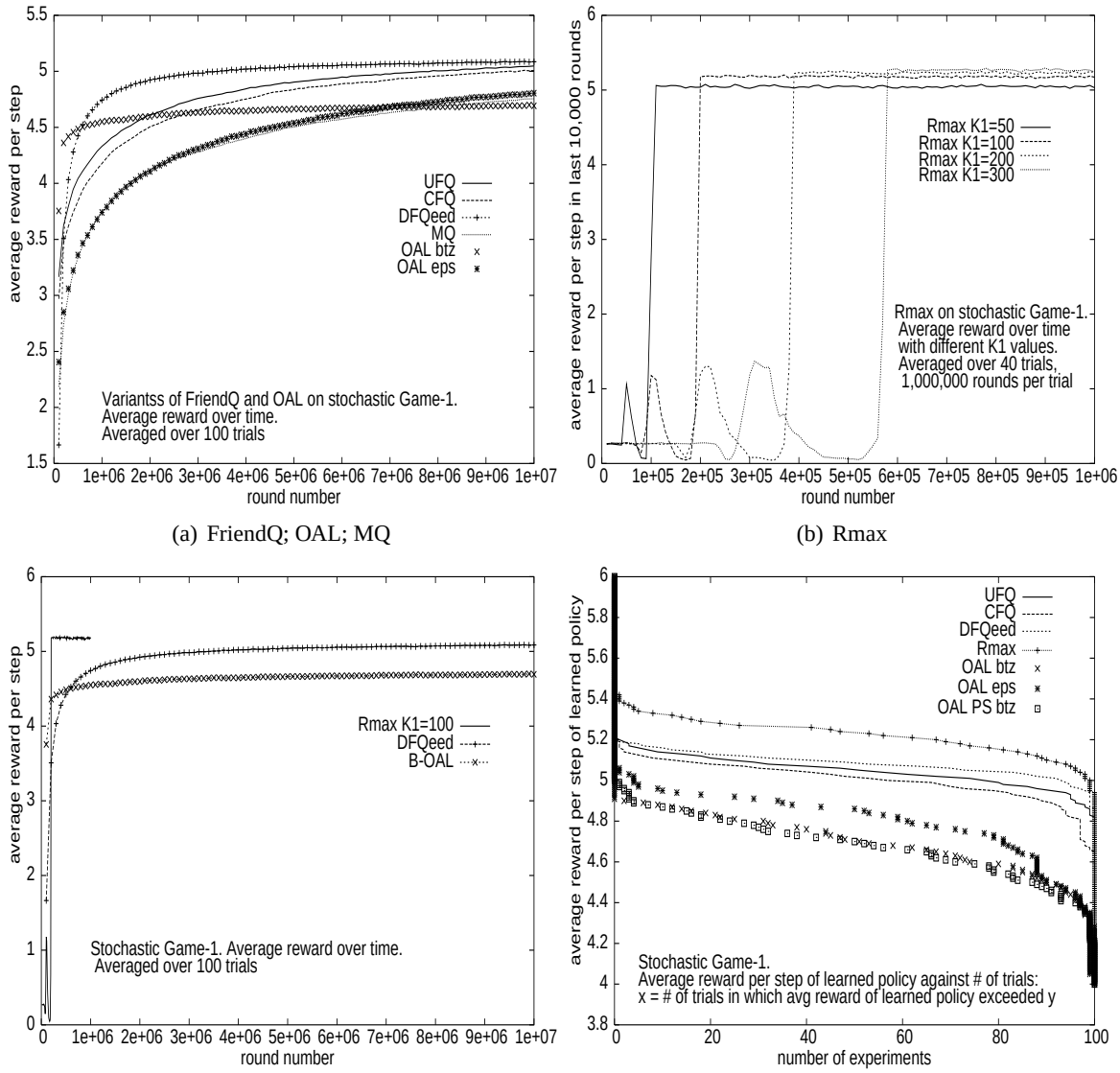
Stochastic Mode

Figure 4 presents the results for the stochastic mode. As expected, due to the stochastic effects of actions, the value of the optimal policy decreases, and more importantly, the learning algorithms require more trials to converge. By contrast to the deterministic case, MQ performs as well as ε-OAL (Fig. 4a). This improvement is attributable to additional exploration stemming from the stochastic nature of the environment. For the same reason, CFQ performs almost the same as UFQ (Fig. 4a). When we compare the gap between the DFQεed to U/CFQ at the first 10^6 rounds in stochastic mode vs. the deterministic mode we find that the gap is smaller. This difference is due to the fact that the additional early exploration supplied by the exponential decay of ε is redundant in the stochastic case. The slightly higher return gained by DFQεed later on is due to the faster decay of ε. Another interesting difference from the deterministic setting is that initially ε-OAL gains lower return than B-OAL but while B-OAL keeps attaining the same average reward, ε-OAL improves slowly over time and eventually gains a higher average reward than B-OAL. In this case, the slower convergence of the exploration probability to zero enables ε-OAL to "overcome" randomly "bad" exploration in initial learning phases.

Rmax behaves similarly in stochastic and deterministic modes. While the other algorithms achieve only near-optimal return, Rmax attains optimal return (Fig. 4b,c). Rmax's strong exploration bias results in low return until model entries are known. From that point on, Rmax attains an optimal return. The histogram (Fig. 4d) shows that Rmax converges to higher return than the other algorithms not only in the average case but also in the worst case (i.e., almost all runs of Rmax were better than the best runs for the other algorithms). Very low values for $K1$, which mean rough transition probability estimates, are enough for computing near-optimal behavior. Indeed, the exploration vs. exploitation tradeoff is evident even in this simple example. We see how a smaller value of $K1$ leads to faster convergence, but at the cost of slightly smaller average reward.

Overall, it appears that the major issue for the FQ and OAL class of algorithms is exploration. As the space of joint-actions is quite large, there are many relevant options to try. Especially in the deterministic case, the rather standard exploration techniques we used appear to be insufficient. Although stochastic domains naturally lead to more exploration, we can see that the model-free algorithms are sub-optimal. It appears that model-free algorithms—at least in their standard form—have difficulty determining whether certain states were explored sufficiently, and that standard exploration schemes are too crude. Overall, many of the phenomena observed in Game 1 were present in Games 2 and 3. Therefore, in the following experiments, only phenomena not observed in Game 1 will be emphasized.

11. If it is known ahead of time that the environment is deterministic, then $K1$ can be set to 1. Similar locality considerations on stochastic environments can help determine tight bounds on $K1$.



(a) FriendQ; OAL; MQ

(b) Rmax

(c) Rmax; FriendQ; OAL

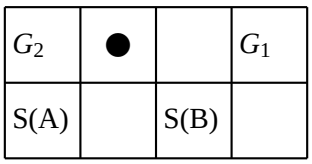
(d) Average reward of learned policies per number of trials

Figure 4: Game 1 – Average reward per step and learned policies per number of trials in stochastic mode. Subfigures (a) and (c) present all 10^7 rounds, while (b) presents the first 10^6 rounds.

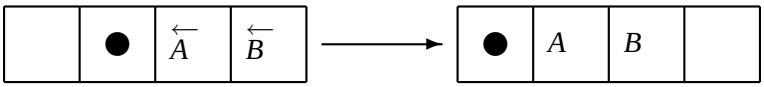
3.5.2 GAME 2

This game was designed to minimize the effects of equilibrium selection, to show how GLIELPs may keep agents exploiting suboptimal possibilities, and to emphasize the importance of coordinated exploration. The game has four goal states and one optimal equilibrium. The game is depicted in Figure 5(a). It consists of an additional element, an object that can be moved by the agents. The agents can move in four directions or stay in place. They can push the object by standing to its right(left) and moving left(right) and pull the object by standing to its right(left) and

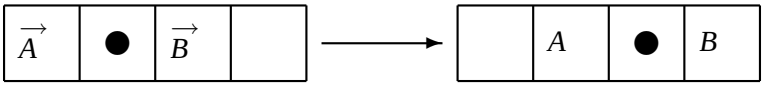
moving right(left). However, the object is too heavy for one agent and requires cooperation of the two agents to be moved. The manner by which the object is moved is depicted in Figure 5(b). Note that the push/pull effect is a by-product of the agents' moves. Thus, in stochastic mode, what determines if the action is push or pull is not the chosen action but its actual effect.



(a) Initial state and Goal states.



(b1) Moving the object by pushing simultaneously.
(Agents' order does not matter).



(b2) Moving the object by pushing and pulling simultaneously.
(Agents' order does not matter).

Figure 5: Game 2

The agents' goal is to move the object into one of the upper corners of the grid, at which point the game is reset to its initial state. Moving the object to the upper right (G_1) or left (G_2) corner yields a reward of 80 and 27, respectively. The optimal behavior under deterministic mode is to move the object to G_1 in 8 steps. The average reward per step of an optimal strategy under deterministic mode is 10, and the discounted return is ~ 465. The second-best strategy is moving the object to G_2 in 4 steps, with an a.r.p.s. of 9 and discounted return of ~ 440. In stochastic mode, the optimal policy may stochastically lead to one of the goal positions. The a.r.p.s. of the optimal policy in stochastic mode is ~ 3.8. The underlying CISG contains 164 states. Algorithms were executed for 3×10^7 rounds.

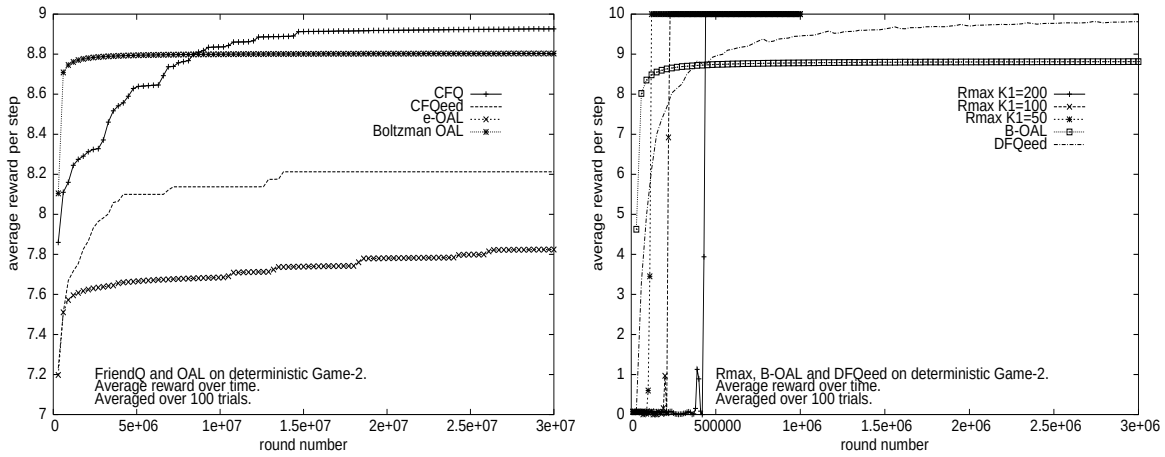
Deterministic Mode

Table 3 classifies the number of trials (of 100 per algorithm) according to the algorithms and learned policies. Figure 6 shows the a.r.p.s. over time obtained by the different algorithms.

The main reasons for the sub-optimal performance of OAL and CFQ in this game are: (i) Random exploration has a greater chance of reaching G_2 than G_1. Discovering G_2 before G_1 further reduces the chance of visiting G_1 because of the increasing bias toward exploitation. (ii) Exploration of the CFQ and OAL agents is not coordinated. If reaching G_2 is the current greedy policy, then G_1 will not be visited unless both agents explore simultaneously. Game 1 demonstrated an advantage of exponential decay of the ε-greedy exploration probability over polynomial decay of this probability. Game 2 demonstrates an opposite phenomenon, Fig. 6a and Table 3 show that CFQ does better with polynomial decay of ε than with exponential decay. This result stems from finite exploration supplied by exponential decay. However, this finite amount of exploration is sufficient

Goal	steps to goal	CFQ	CFQεed	DFQεed	ε-OAL	B-OAL	Rmax
G_1	8			100		1	100
G_2	3	99	65		54	91	
G_2	4	1	35		46	8	

Table 3: Game 2 – Characteristics of the learned policy on a per-trial basis for each algorithm in deterministic mode.



(a) FriendQ and OAL.

(b) Rmax; B-OAL; DFQeed.

Figure 6: Game 2 – Average Reward under deterministic mode. Subfigure (a) presents all 3×10^7 rounds; Subfigure (b) presents first 3×10^6 rounds.

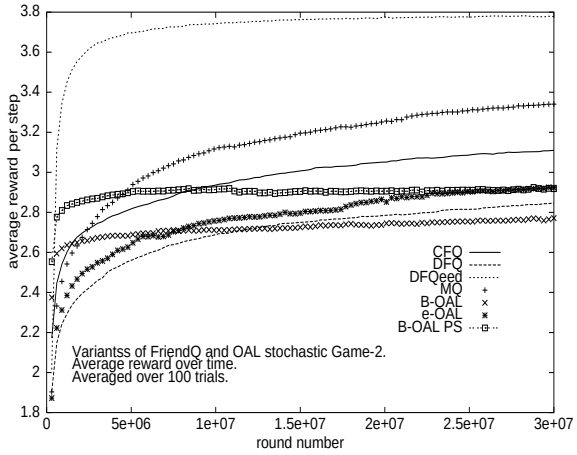
when exploration is coordinated as shown by the learning curve of DFQεed (Fig. 6b) and by Table 3. Furthermore, DFQεed converges to optimal greedy behavior while both CFQ variants do not.

Stochastic Mode

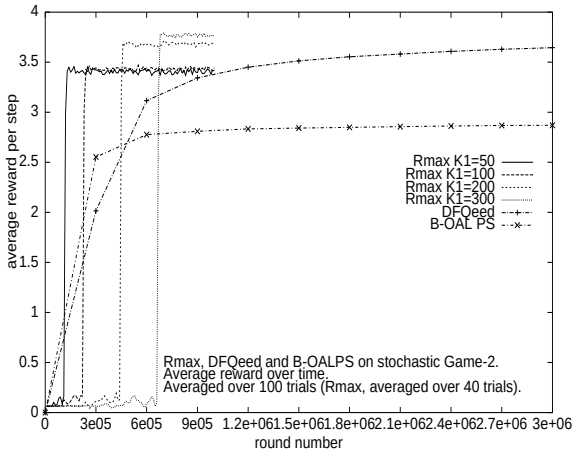
Figure 7 presents statistics for the stochastic mode. It exhibits two interesting phenomena that have not been observed in the previous experiments. One is that, in contrast to previous results, ModelQ outperforms ε-OAL (Fig. 7a,c). Since the only difference between ε-OAL and ModelQ is the greedy action selection method, a reasonable explanation is that BAP (OAL's action selection mechanism) delays behavioral change that should follow Q-value updates (which in turn may delay learning of Q-values). This outcome is because BAP plays a best response to the strategy implied by the other agent's past plays. Since both agents react to each other's past plays using BAP, it may take long to converge to a new NE when the optimal joint actions are changed. The second phenomenon is that Rmax requires larger values of $K1$ to converge to optimal behavior (Fig. 7b). This finding can be explained by the fact that the optimal behavior involves longer cycles of state transitions and hence the model has to be more accurate.

As in Game 1, we see that exploration strategies have a great impact on the ability of different algorithms to converge. In this respect, Game 2 highlights the need for coordinated exploration.

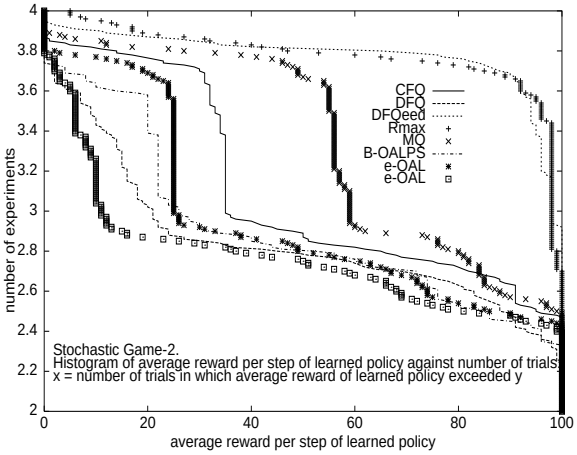
Thus, in cooperative multi-agent systems, we face the standard problem, clearly visible in Game 1, of ensuring sufficient exploration, but we need to ensure that this exploration is effective by coordinating exploratory moves of different agents.



(a) FriendQ and OAL



(b) Rmax; B-OALPS; DFQeed



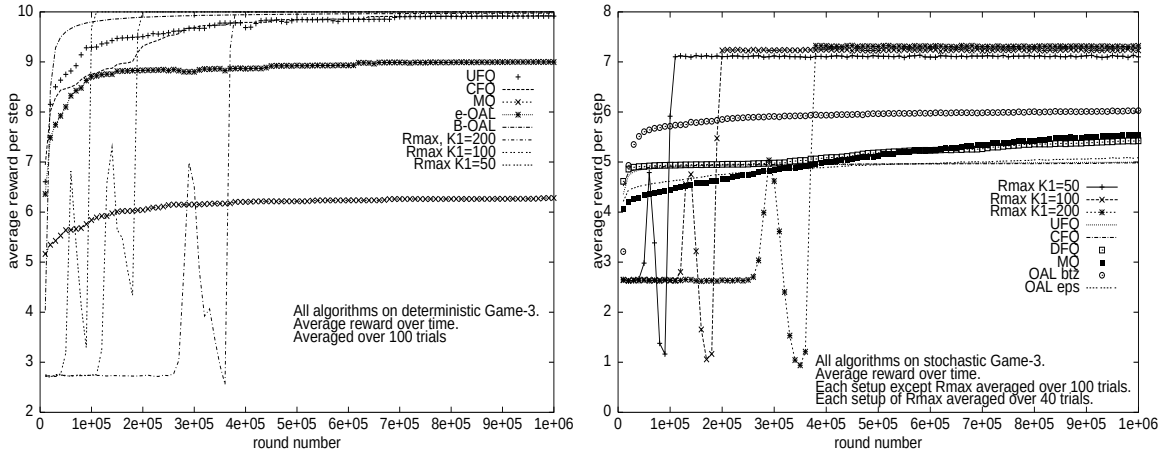
(c) Average Reward of Learned Policies per Number of Trials

Figure 7: Game 2 – Average reward per step and learned policies per number of trials under stochastic mode. (a) presents all 3×10^7 rounds; (b) presents first 3×10^6 rounds.

3.5.3 GAME 3

In the previous games, one had to explore considerable parts of the state space in order to construct good policies. This game is characterized by a maximum return attainable by staying in a small local set of states anywhere on the state graph. The initial position of the agents within a 3×3 grid is random. They are rewarded for reaching a position in which their locations are adjacent. If this position is attained by unaltered positions of both agents, the reward is 5. If movement is involved, the reward is 10. Algorithms were executed for 10^6 rounds.

As opposed to previous experiments, in deterministic mode, B-OAL and the FriendQ variants converged faster to optimal (greedy) behavior than Rmax (Fig. 8a). Rmax explores the whole model before it starts exploiting while FriendQ's and OAL's choice of greedy actions is optimal long before good estimates of all Q-values are attained. However, in stochastic mode the GLIELPs no longer have this advantage since stochastic transitions do not enable the agents to concentrate on exploiting a local set of states (Fig. 8b).



(a) deterministic mode (b) stochastic mode

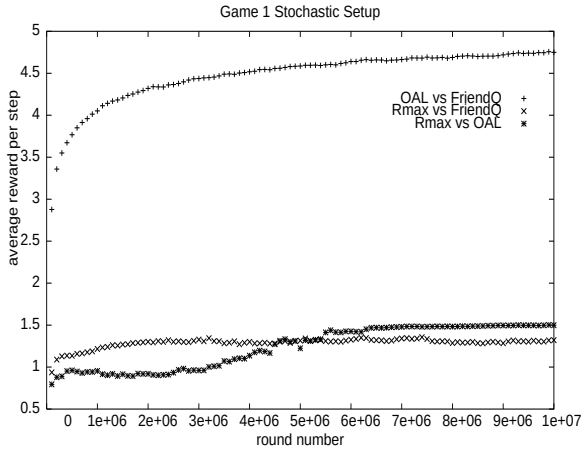
Figure 8: Game 3 – average reward per step under deterministic and stochastic modes.

3.5.4 HETEROGENEOUS PLAYERS

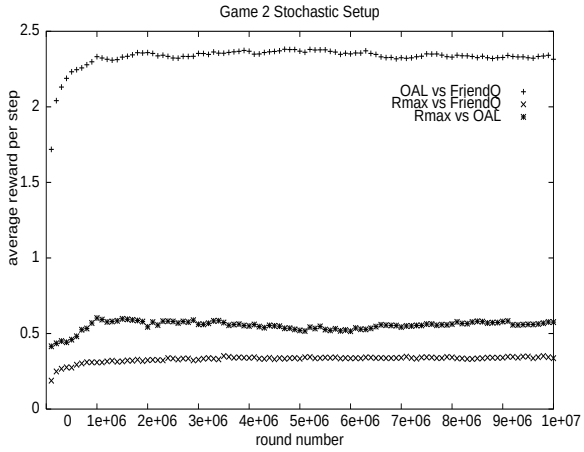
A CISG is most naturally viewed as a model of a distributed stochastic system. As such, it is natural to have in mind a view of a system's designer, and one would expect such a designer to equip the players with identical algorithms. However, CISGs arise also when self-interested agents need to coordinate, typically on the use of some resource, where coordination is beneficial to all parties involved. Examples include which side of the road to travel on, the meaning attached to a symbol, etc. Thus, it is natural to ask how the algorithms tested fare in the context of other algorithms. We reran the above experiments with pairs of different algorithms. The results, presented in Figure 9 are, quite uniform (similar performance is observed in the deterministic games). The top performance, and as is clearly visible, by a wide margin, was obtained by OAL+FriendQ. Pairs containing Rmax performed much worse, with Rmax+OAL typically fairing slightly better than Rmax+FriendQ. In fact, comparing the results to the homogeneous case, the OAL+FriendQ combination performed almost optimally in Game 1: 4.7 vs. 5. In Game 2 it obtained 2.3 vs. 3.8, and in Game 3 it achieved 5.85 vs. 7.2.[12] And while Rmax and, to a lesser extent, FriendQ do better against their own kind, OAL does better against FriendQ. It may be the case that for equilibrium selection, OAL's mechanism works best when one agent "insists" more on a particular equilibrium, thus more quickly breaking up symmetries.

These results might be interpreted as an indication of the "rigidity" of each of the algorithms. FriendQ is the simplest of the three algorithms, it makes no internal assumptions about its partners

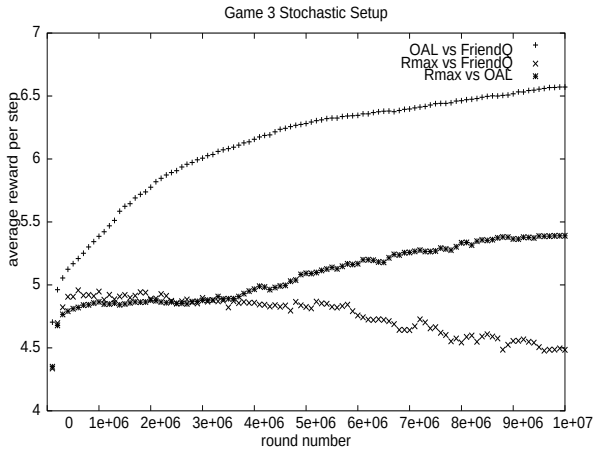
12. The version of FriendQ used was CFQ with replacing traces. The OAL version used ε-greedy exploration.



(a) Game 1 stochastic mode



(b) Game 2 stochastic mode



(c) Game 3 stochastic mode

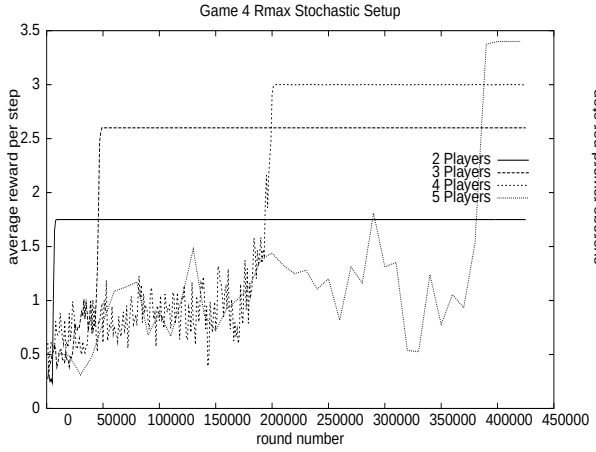
Figure 9: Heterogeneous play games 1-3.

and simply adapts. Rmax is at the other extreme, it strongly relies on the behavior of its partners in order to systematically explore and then exploit. OAL is somewhere in between. It does have a sophisticated mechanism for selecting among different equilibria, but this mechanism is stochastic and can handle noise, and is based on fictitious play, which is a mechanism that adapts to the empirical behavior of the other agents. Thus, one would expect Rmax to fail when its assumptions are not met, as its implicit coordination mechanism is based upon them. In contrast, FriendQ and OAL, which make weak internal assumptions about their peers, should work well, especially when their opponent shows some flexibility and adaptivity.

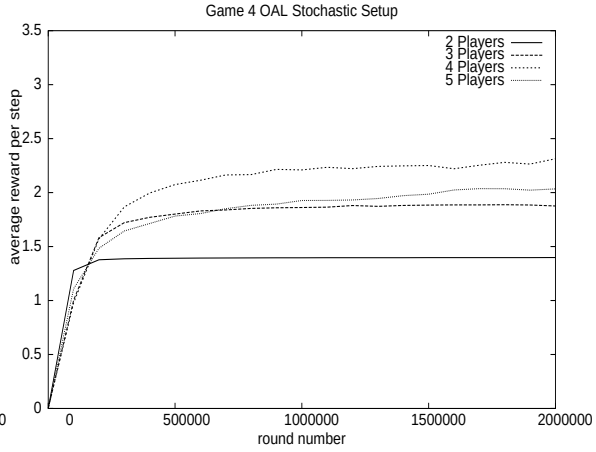
3.5.5 $n > 2$ PLAYERS

So far, we considered only two-player games. The reason is practical: Experiments conducted on large state spaces take long to execute. It is especially true for Rmax which must solve the MDP each time the model changes. This effort grows with the state-space, and the state-space grows

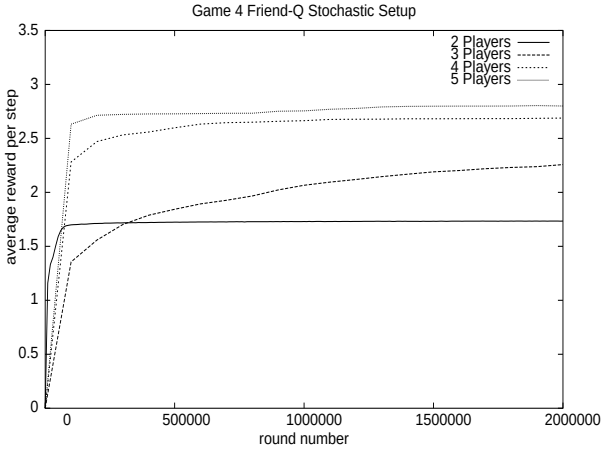
exponentially with the number of players. Thus, to get an idea of how these algorithms fare with a large number of players, we devised a simpler, fourth game in which we could run experiment with up to 5 players. This is a simple linear grid with 5 positions. Players can move to the left and the right. When two players attempt to move to the same position, the result is with probability 1/3 none move, and with probability 1/3 each one of the players makes the move and the other stays in place. In the initial state, player i is in position $5-i$. The goal position of each player is i. Generally, the reward at each state is the number of players located at their goal positions. However, when all players are in their goal position, they receive a reward of 3 *times* the number of players, at which point all players transition automatically to the initial position.



(a) Game 4 R-max



(b) Game 4 OAL



(c) Game 4 Friend-Q

Figure 10: Results for 2-5 players on game 4.

The results are presented in Figure 10. Note the difference in scale for the X-axis for OAL and FriendQ, which is intended to show that the suboptimal a.r.p.s. to which they converge does not increase even when we look at millions of steps. As in the previous section, we used the CFQ version of FriendQ with replacing traces, ε-greedy exploration for OAL, and $K_1 = 100$ for Rmax.

For all algorithms, the value is greater as the number of players increases due to the game's reward definition, which is sensitive to the number of players. All algorithms converge quickly on this game for all number of players. However, the values they converge to differ. Among the three algorithms, Rmax converges to the highest average per step reward, FriendQ is next, and OAL is last. The relative performance is consistent with the performance displayed in the three earlier two-player games. This finding is a reasonable indication that the relative performance of these algorithms is qualitatively similar regardless of the number of players, at least for small player sets.

3.5.6 SUMMARY

Section 3.5 presents an experimental study of three fundamentally different algorithm families for learning in CISGs. The results illustrate the strengths and weaknesses of different aspects of these algorithms in different settings, highlighting the accentuated importance of effective exploration, which is enabled in this class of games only by coordinated behavior, the advantage of deterministic behavior for attaining such coordinated behavior, and the benefits of propagation of information.

Each of the experimental domains emphasizes different aspects of the learning task in CISGs. The results show that the parameters of OAL and FriendQ are very sensitive to environmental topology and dynamic. Exploration and coordination strategies suitable for one environment may be very inefficient in other environments. Rmax, on the other hand, is stable in this respect. It has a single parameter, $K1$, that has to be preset. Its convergence time depends linearly on $K1$ (and the size of the state-action space) and it turns out that Rmax converges to near optimal behavior using values of $K1$ that achieve faster convergence than that of OAL and FriendQ. However, the convergence dynamics of Rmax does not suit tasks in which the agents must attain some value during the learning period, because during its exploration phase, Rmax is indifferent to rewards lower than R_{max}. However, Rmax is also the simplest algorithm, and thus it is easy to alter it, for instance, to obtain satisficing behavior, for example, by lowering the value of R_{max} in the model, or by starting with a moderate value and then increasing it as better values are observed. Overall, when we control the algorithm of all agents in the system, Rmax seems to be the best alternative—it converges quickly to values higher than those of OAL and FriendQ, it does not seem to be sensitive to an increase in the number of players, except through its effect on the state space, and most importantly, it has very simple exploration strategy. As we saw in games 1 and 2, the choice of exploration strategy has much influence on the results of FriendQ and OAL, and the precise choice is sensitive to the nature of the game, number of equilibria and their nature. Thus, the simple exploration behavior of Rmax and the potential to alter it in various transparent ways is a clear benefit. Yet, when we may need to coordinate with other players with unknown coordination mechanisms, or if our underlying state space is too big for repeated value computations, FriendQ seems to offer the best choice.

4. Learning in FSSGs

In two-player *Zero Sum* SGs (ZSSGs), the players' payoffs sum up to zero at every entry. That is, $[R(s,a^1,a^2)]_1 = -[R(s,a^1,a^2)]_2$ for every $s \in S$, $a^1 \in A_1$ and $a^2 \in A_2$. Such payoffs indicate that the agents' interests completely conflict. A ZSSG can be modeled with a single payoff function $R'(s,a^1,a^2) = [R(s,a^1,a^2)]_1$ by redefining Player's 2 objective as to minimize the IHDR (infinite horizon discounted reward). For the rest of this section, it is assumed that Player 1 is the maximizer and Player 2 is the minimizer the payoff function R. Let $V(s,\pi_1,\pi_2)$ denote the expected IHDR for starting at state s and playing the profile (π_1,π_2) of stationary mixed policies there-

after, and $V(\pi_1,\pi_2) = (V(1,\pi_1,\pi_2),...,V(|S|,\pi_1,\pi_2))$. Since the best response in a ZSSG is also the worst for the opponent, ZSSGs have a unique NE value. To see this, assume by negation that $V(s,\pi_1,\pi_2) > V(s,\mu_1,\mu_2)$ and that both (π_1,π_2) and (μ_1,μ_2) are NE. Since $\pi_2 \in BR(\pi_1)$, it follows that $V(s,\pi_1,\mu_2) \geq V(s,\pi_1,\pi_2) > V(s,\mu_1,\mu_2)$, which contradicts $\mu_1 \in BR(\mu_2)$. The value of a policy π may be defined as $V(\pi,BR(\pi))$. In ZSSGs, this definition coincides with that of a NE (any pair of optimal policies is a NE and vice versa).

Consequently, the state-value function $V(s)$ is redefined to be the expected IHDR under a profile of optimal policies and $Q(s,a_1,a_2)$ the expected IHDR for taking joint action (a_1,a_2) in state s and continuing according to a NE thereafter. For any stationary strategy profile (π_1,π_2) in a ZSSG G, $(\pi_1(s),\pi_2(s))$ is a NE for the matrix games defined by $[Q(s,a^1,a^2)]_{a^1\in A_1,a^2\in A_2}$ for all $s \in S$ if and only if (π_1,π_2) is a NE for G and the NE values for the matrix games correspond to the state values $V(s,\pi_1,\pi_2)$ (Filar and Vrieze, 1997). Thus, the Bellman optimality equations can be rewritten for ZSSGs as

$$\begin{aligned} Q^*(s,a^1,a^2) &= R(s,a^1,a^2) + \gamma \sum_{s'} T(s,a^1,a^2,s')V^*(s') \\ V^*(s) &= \sum_{a^1\in A_1,a^2\in A_2} \pi_1(a^1)\pi_2(a^2)Q(s,a^1,a^2) \end{aligned} \quad (1)$$

where (π_1,π_2) is a NE for the matrix game defined by the Q-values in state s. Given a method that computes NE for zero sum matrix games, Equation (1) can be used as an iterative approximation rule to compute the Q-values (Littman, 1994) and given the Q-values an optimal policy can be derived. The NE policies for a zero sum matrix game $M = [r(a_i,b_j)]_{i=1,j=1}^{k,l}$ are the solutions to the linear program that maximizes v under the constraints (Filar and Vrieze, 1997)

$$\left\{ \sum_{i=1}^{k} \pi(a_i) r(a_i,b_j) \geq v \mid j \in \{1,...,l\} \right\}.$$

In the following sections, this linear program is abbreviated as:

$$v = \max_{\pi\in PD(A)} \min_{b\in B} \sum_{a\in A} \pi(a) r(a,b).$$

If SG G is obtained from ZSSG G' by adding a constant c to all payoffs of **both** players, then $V_i^G(\pi_1,\pi_2) = V_i^{G'}(\pi_1,\pi_2) + c/(1-\gamma)$ for any policy profile (π_1,π_2) and the strategic properties of the game are unchanged. G is referred to as a Fixed Sum Stochastic Game (FSSG). The adversarial nature of FSSGs calls for agents that perform well not only in self play but also in *heterogeneous play*, namely when engaged by agents that employ different learning algorithms. Under this setting, the exploration/exploitation tradeoff wears a new guise as attempted exploration and exploitation may be interfered by the opponent.

This section compares three algorithms for learning in FSSGs: FoeQ (Littman, 1994, 2001), WoLF (Bowling and Veloso, 2002) and Rmax (Brafman and Tennenholtz, 2002). They were selected because they represent different approaches to the exploration/exploitation tradeoff and to information propagation while providing some theoretical guarantees. Specifically, Rmax and FoeQ converge to a NE in FSSGs in self-play, while WoLF is known to converge in 2 player, 2 action, repeated games.

4.1 FoeQ

FoeQ (a.k.a MinimaxQ) (Littman, 1994, 2001) extends Q-learning into FSSGs by using a sample backup learning rule based on Equation 1 (Littman, 1994). After taking a joint action (a,b) in state s at time t and reaching state s' with reward r_{cur}, the agent updates the Q-value of $\langle s,(a,b)\rangle$ by

$$Q_t(s,a,b) \leftarrow (1-\alpha_t)Q_{t-1}(s,a,b) + \alpha_t \left(r_{cur} + \gamma \max_{\pi \in PD(A)} \min_{b' \in B} \sum_{a' \in A} \pi(a') Q(s',a',b') \right).$$

Q_t converges in the limit to Q^* under the standard Q-learning conditions stated in Section 3.1. Also, for similar reasons to those stated in Section 3.1, FoeQ is executed with an ε-greedy learning policy.

4.2 WoLF

WoLF (Bowling and Veloso, 2002) is designed to converge to a best response rather than a NE. WoLF does not explicitly consider an adversary. It applies the standard single-agent Q-learning rule to approximate Q-values of private actions and uses hill climbing to update its mixed policy. That is, the policy is improved by increasing the probability of selecting a greedy action according to a policy learning rate δ (which is distinct from the Q-value learning rate α), enabling mixed policies. The uniqueness of WoLF is in using a *variable* policy learning rate according to the "Win or Learn Fast" (hence WoLF) principle: if the expected return of the current policy given the current Q-values is below (above) a certain threshold then a high, δ_l (low, δ_w), learning rate is set. A good threshold would be the NE value of the game because if the player is receiving less than its value, its likely playing a sub-optimal strategy, whereas if it receives more than the NE value, the other players must be playing sub-optimally. Since the NE value is unknown, it is approximated by the expected return of the average policy (averaged over the history of the game) given the current Q-values. The motivation for the WoLF variable policy learning rate is to enable convergence to a NE. Indeed, Bowling and Veloso (2002) show that gradient ascent with WoLF is guaranteed to converge to a NE in self play on two-player, two-action, repeated matrix-games, while gradient ascent without a variable learning rate is shown not to. Furthermore, they provide empirical results on FSSGs in which WoLF converges to NE in self play. WoLF, as single agent Q-learning, is guaranteed to converge in the limit to a best response under the standard conditions and given that the opponent(s) converge to stationary policies.

4.3 Rmax

Section 3.3 describes the Rmax algorithm in the context of MDPs. The same algorithm is applicable to FSSGs with the only difference that joint actions are considered and optimal policies with respect to the fictitious model are computed according to (1). As mentioned in Section 3.3, Rmax always behaves optimally with respect to an approximated, initially optimistic, model M' of the real model M. Since unknown entries are modeled in an attractive manner in M', Rmax has a strong bias to explore. Seeing that the optimal policy maximizes return against the worst opponent, if the opponent prevents Rmax from visiting unknown entries then Rmax attains near-optimal return because the known entries are accurately modeled. Thus, Rmax is guaranteed to either attain near optimal return in the real model M or, with sufficiently high probability, visit unknown entries (Brafman and Tennenholtz, 2002). This property assures that Rmax will attain near-optimal average reward after a polynomial number of steps in FSSGs as well as in MDPs.

As in the CISGs experiments, $K1$, the number of visits required to declare an entry known, is treated as a parameter that has to be preset and R_{max} is not assumed to be known. Instead, R_{max} is initialized to some positive value and updated online to be twice the highest reward encountered so far.

4.4 Discussion of Algorithms

The shortcoming of GLIELPs discussed in Section 3.4, namely the possibility of untimely greediness, applies also to FoeQ and WoLF in fixed sum environments and may be further exploited by an informed adversary. Furthermore, FoeQ and WoLF do not reason about how the opponent affects exploration. Thus, attempted exploration may result, depending on the opponent's action choice, in joint actions that are of low informative and materialistic value. FoeQ's and WoLF's single step backups and possible premature decrease of the Q-learning rates may cause poor use of new experience.

WoLF compensates for the above limitations by the following properties: (i) Hill climbing adjustment of the policy for enhanced exploration. Specifically, this exploration is regulated by the variable policy learning rate to explore more while "winning". The gradual policy update also prevents formation of big gaps between Q-values of different entries and thus contributes to both fast adjustment to changes in the adversary's behavior and reduction of the effect of untimely greediness. (ii) WoLF's greedy policy is a best response rather than a NE. This fact results in high payoffs during learning, fast growth of Q-values and hence fast convergence. (iii) WoLF does not explicitly model the opponent. It maintains Q-values for the small action space of private actions resulting in faster propagation of state-action values.

A major conceptual difference between WoLF and both Rmax and FoeQ is the target of learning, which also implies the definition of greediness during learning. Rmax's and FoeQ's greedy policies are NE policies. Playing a NE policy is the best strategy against a rational opponent. It also makes sense even if the adversary does not play a BR since it ensures at least the value of the game. However, playing a NE policy w.r.t. a non-accurate model/Q-values during learning makes the hidden assumption that the opponent is not only rational but also acts according to the same model/Q-values. Under heterogeneous play, this assumption is not valid and may result in low payoffs during learning. FoeQ typically maintains pessimistic Q-values during learning. The resulting (greedy) NE learning policy will attempt to avoid entries that are not well known since they seem unprofitable. Thus, FoeQ's greedy action choice may have low informative and materialistic value when engaged in heterogeneous play. For model free algorithms, fast convergence depends on high payoffs during learning. Rmax does not distinguish exploration from exploitation and guarantees to either exploit or explore independent of the adversary's actions. Thus, low payoffs during learning are traded for faster convergence to a NE policy. However, Rmax is biased to explore and may play exploratory actions even when it "knows" a submodel in which the value is attainable. WoLF pursues the best response policy during learning. This strategy is efficient against adversaries that converge to stationary policies. However, an adversary that knows WoLF's strategy may play a "decoy" policy until WoLF's learning is slow and then switch to a best response.

Notwithstanding formal results (Even-Dar and Mansour, 2003), parameter tuning is still a task that requires expert experience and intuition. In this respect, WoLF is the most complicated among the three algorithms. On top of the parameters for decaying exploration and Q learning rates, which also appear in FoeQ, it involves presetting the decay rate of the policy-learning rates and the relation

between the two policy-learning rates δ_w and δ_l. Rmax is the simplest to tune among the three with a single parameter, $K1$, the number of visits to declare an entry "known".

Finally, We note that our discussion of convergence rates and the especially our experimental evaluation focuses on the number of time steps,or multi-agent encounters rather than CPU-time. Although the algorithms we evaluated are all considered practical for online reinforcement learning, it should be noted that WoLF requires considerably less computation than FoeQ or Rmax since it does not involve linear programming computations of equilibria.

4.5 Experimental Results & Analysis

This section describes experimental results on three 2-agent fixed-sum grid games. The games were designed to evaluate the effects of exploration, information propagation, action selection and other methods, on the performance of FoeQ, WoLF and Rmax in different environments. The algorithms were tested in both self play and heterogeneous play. The available actions, indexing of the grid, transition probabilities for the deterministic and stochastic modes, discount factor etcȧre the same as in the CISG experiments. The process of adjusting the parameters was also similar to the CISG experiments and was conducted on the "deterministic 3×3 Wall game" (see below).

The following parameter settings were used for testing:

FoeQ

Exploration: ε-greedy, $\varepsilon_t \leftarrow \max\left\{0.99999^{count_t}, 1/count_t^{0.5000001}\right\}$ where $count_t$ is the number of exploratory steps taken by time t.

Learning rate: $\alpha_{s,a} \leftarrow \max\left\{0.99908^{n(s,a)}, 1/n(s,a)^{0.75}\right\}$ where $n(s,a)$ is the number of times action a was taken in state s.

Q-values were initialized to 0.

WoLF

Exploration: ε-greedy, $\varepsilon_t \leftarrow \max\left\{0.5000001^{count_t}, 1/count_t^{0.5000001}\right\}$.

Q Learning rate: $\alpha_{s,a} \leftarrow \max\left\{0.95^{n(s,a)}, 1/n(s,a)^{0.0.5000001}\right\}$.

Policy Learning rate: $\delta_l = 0.7 \times \alpha_{s,a_g}, \quad \delta_w = 0.175 \times \alpha_{s,a_g}$ where a_g is a greedy action in the current state s.

Rmax

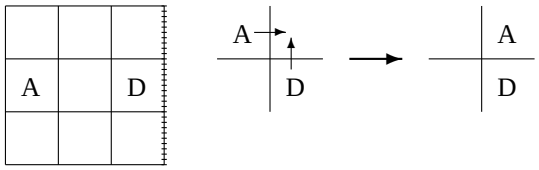
Sampling: values of 50, 100 and 200 for $K1$ (visits to mark an entry known) were tested.

Accuracy of Policy Iteration: Offline Policy Iteration was halted when the difference between two successive approximations was less than 0.001

4.5.1 3×3 WALL GAME

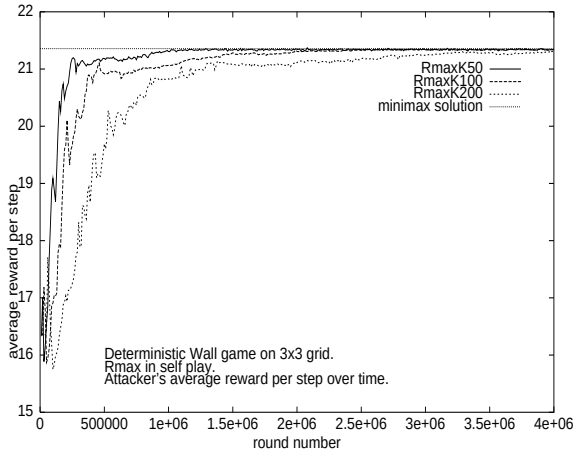
In this 3×3 grid game, one player, A, is an Attacker and the other, D, is a Defender. Figure 11a depicts the initial position of the game. A's goal is to reach the rightmost column of the grid. If both players try to enter the same square or to enter each other's current positions (that is, switch places) then their locations are unchanged. The only exception to this rule is when the players are in diagonally adjacent squares—in this case A moves and D's position is unchanged (Fig. 11b), so that

the attacker has a slight advantage. The fixed sum of the game is 40. When A reaches the rightmost column of the grid, it receives a reward of 40, D receives a reward of 0 and the players are reset in their initial positions. For any other move, A is rewarded by 15 and D is rewarded by 25. The game was played under deterministic transition probabilities. Every experimental trial was over 4×10^6 rounds. The minimax a.r.p.s. for the Attacker is ~ 21.36.

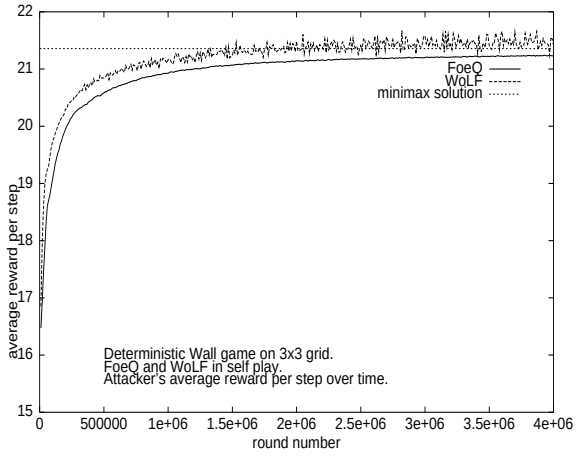


(a) Initial position (b) Attacker passes defender

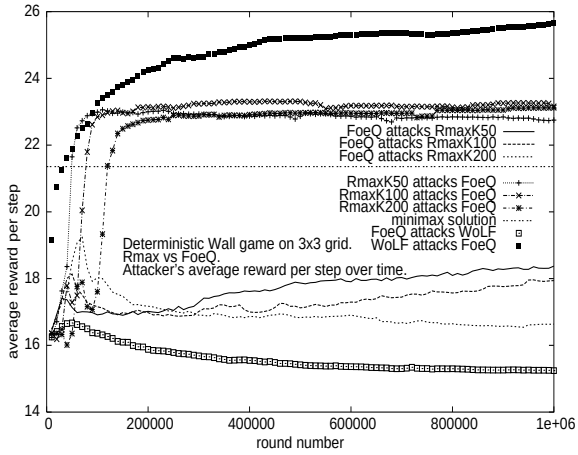
Figure 11: 3x3 wall game



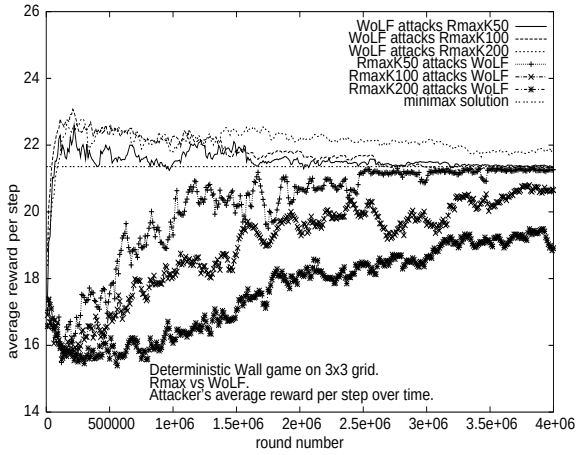
(a) Rmax in self play



(b) FoeQ and WoLF in self play



(c) Rmax vs FoeQ and WoLF vs FoeQ (**only first** 10^6 **rounds**)



(d) Rmax vs WoLF

Figure 12: 3x3 wall game – average reward per step

			Attacker's policy							Defender's policy						
A	*D*	*K*1	u	l	d	r	s	Q-val	KE	u	l	d	r	s	Q-val	KE
Opt	**Opt**		**.514**	**.0**	**.0**	**.486**	**.0**	**1053**	**1200**	**.0**	**.383**	**.0**	**.617**	**.0**	**947**	**1200**
RX	RX	50	.514	.0	.0	.486	.0	—	838	.0	.383	.0	.617	.0	—	838
RX	RX	100	.514	.0	.0	.486	.0	—	833	.0	.383	.0	.617	.0	—	833
RX	RX	200	.515	.0	.0	.485	.0	—	831	.0	.385	.0	.615	.0	—	831
FQ	FQ		.415	.0	.297	.288	.0	966	—	.0	.383	.0	.327	.290	947	—
WF	WF		.279	.006	.279	.432	.004	1095	—	.067	.309	.022	.278	.324	966	—
RX	FQ	50	.213	.0	.181	.606	.0	—	391	.087	.022	.0	.525	.366	265	—
RX	FQ	100	.382	.0	.225	.393	.0	—	295	.054	.066	.068	.597	.215	452	—
RX	FQ	200	.154	.0	.241	.605	.0	—	249	.074	.023	.009	.630	.264	715	—
FQ	RX	50	.040	.094	.0	.727	.139	204	—	.026	.484	.017	.473	.0	—	638
FQ	RX	100	.018	.084	.033	.800	.065	338	—	.048	.683	.075	.194	.0	—	516
FQ	RX	200	.009	.115	.005	.847	.025	517	—	.033	.810	.037	.120	.0	—	391
RX	WF	50	.262	.0	.262	.476	.0	—	583	.06	.147	.013	.334	.446	958	—
RX	WF	100	.388	.0	.170	.442	.0	—	579	.109	.043	.027	.419	.402	997	—
RX	WF	200	.442	.025	.348	.185	.0	—	510	.253	.031	.142	.237	.337	1055	—
WF	RX	50	.238	.0	.185	.577	.0	1061	—	.0	.384	.004	.426	.186	—	487
WF	RX	100	.214	.0	.231	.555	.0	1060	—	.002	.371	.0	.532	.095	—	411
WF	RX	200	.314	.0	.303	.378	.005	1082	—	.026	.401	.010	.524	.039	—	365

Table 4: 3×3 wall game – The first row reports the action probabilities of a NE policy profile in the initial state, the Q-values for action $\langle stand, stand \rangle$ in the initial state and the number of entries in the game. The next rows classify the average learned policies in the initial state, the average learned Q-values for actions $\langle stand, stand \rangle / \langle stand \rangle$ by the FoeQ/WoLF players, respectively, in the initial state and the average number of known entries by Rmax, after 4×10^6 rounds, according to the players' types. RX, FQ and WF are abbreviations for Rmax, FoeQ and WoLF respectively. The first column, titled *A*, provides the Attacker's type. The second column, titled *D*, provides the Defender's type. The third column, titled $K1$, states the value of Rmax's $K1$ parameter. The columns titled *u*, *l*, *d*, *r*, *s* specify the probabilities for actions *up, left, down, right,* and *stand*, respectively, according to the learned policies. The columns titled *Q-val* and *KE* specify the learned Q values by FoeQ or WoLF and the number of known entries by Rmax.

Figure 12 presents the a.r.p.s. obtained by the different agents playing the Attacker's role in self and heterogeneous play.[13] Table 4 classifies some significant variables of the average state of the learning algorithms after 4×10^6 rounds according to the players' types.

Self Play

In self play, all algorithms converge to minimax or almost minimax values (Fig. 12a,b). FoeQ converges to within 0.5 of the minimax value within the first 10^6 rounds and from then on improves very slowly because of its increasing bias to exploit combined with its decreasing learning rates. The FoeQ Defender learns correct Q-values and an optimal policy while the Attacker learns a rough estimation of the Q-values and a suboptimal policy (Table 4).[14] WoLF converges to the minimax value within 1.5×10^6 rounds and then oscillates around this value while the players keep updating

13. In figures where the behavior does not change after a certain point, we show only the initial phases. For example, in Figure 12c).

14. The first row of Table 4 presents the policies as outputted by a value-iteration solver. It should be noted that there are equivalent optimal policies in the initial state: i. For the Attacker, a probability mass of .514 may be divided in any way between the actions *up* and *down*; ii. For the Defender, the actions *right* and *stand* are equivalent since it is next to the right border of the grid, and for the Attacker, *left* and *stand* are equivalent.

their best responses to each other (Fig. 12b). WoLF learns almost optimal policies and Q-values (Table 4). Three main differences make WoLF more robust than FoeQ: first, it maintains a considerably smaller state-action space since it considers only private actions. This difference results in more efficient back propagation of the Q-values. Second, its learning policy is a best response policy rather than an equilibrium policy. For this reason, it collects higher rewards during early phases of learning and in turn the Q-values converge faster. And third, the hill climbing policy updates combined with the variable learning rate serve as an additional exploration mechanism. As long as the WoLF players have not converged to equilibrium, an increased policy learning rate will always be used by one of the players. As in the CISG case, Rmax's learning period depends almost linearly on $K1$. With $K1 = 50$, Rmax converges to the minimax value within 10^6 rounds (Fig. 12a). Unlike the CISG case, Rmax converges to optimal policies before all entries become *known* (Table 4), thus the unknown entries will not be (unnecessarily) further explored.

In self play, the identical exploration and exploitation techniques of both players gives rise to efficient *joint* exploration and hence to fast convergence to policies that are close or equal to the minimax policies. In contrast, when the opponents employ different learning algorithms, joint exploration is impeded.

Heterogeneous Players

Figure 12c depicts the average learning curves for plays of FoeQ against Rmax. The curves start at ~ 16.25, which is the a.r.p.s. for the Attacker when random policies are played. The learning curve for the FoeQ Attacker may be divided into three phases: in the first 50,000 to 100,000 rounds (depending on $K1$), FoeQ's a.r.p.s. increases rapidly, then drops back down to ~ 17 and changes very slowly thereafter. During the first phase, Rmax plays exploratory policies that enable FoeQ to reach its goal states by playing suboptimal policies. As a result, FoeQ propagates Q-values of entries that are not frequently visited by the NE strategies of the game and constructs a wrong estimate of the strategic structure of the game. During the next phase, Rmax learns improved strategies. At this stage, FoeQ is too biased to exploitation and the learning rates for some entries are too small to overcome the distorted estimation in the first phase. In the third phase, new entries rarely become known because of FoeQ's bias to exploit and its slow learning. For lower values of the $K1$ parameter in Rmax, FoeQ yields lower return in the first phase but recovers faster in the third phase because the first phase is shorter for lower values of $K1$. This property in turn results in lower estimation of the Q-values (Table 4) and hence smaller gaps between the true strategic structure of the game and the strategic structure estimated by FoeQ after the first phase. The smaller gaps are easier to overcome in the third phase. The learning curves and learned values and policies for the opposite mode, Rmax Attacks FoeQ, are a bit less distinct but express similar dynamics. The main advantages of Rmax over FoeQ, expressed in the results, are: An exploration technique that is not time dependent instead of increasing greediness, exhaustive computations instead of single backup per step, and the Rmax learning technique that is guaranteed to either explore or attain return that is at least near the minimax, instead of heuristic exploration. Since, from some early stage, FoeQ mainly attempts to exploit w.r.t. its inaccurate estimation of Q-values and strategic structure and since unknown entries are estimated by FoeQ as having low Q-values, the joint policy does not frequently reach unknown entries but rather yields high rewards for Rmax.[15] Figure 12c also shows the learning curves for

15. It is known that Q-learning with low initial values can behave sub-optimally. Thus, the observed behavior of FoeQ may be a consequence of the fact that Q-values are initialized to 0. However, unless prior information about the possible rewards or their magnitude is available, this appears to be the most unbiased choice.

plays of FoeQ against WoLF. The combination of FoeQ's slow learning, fast "aging" and playing an equilibrium policy w.r.t. its pessimistic estimated Q-values with WoLF's fast learning of a good response produces very poor performance for FoeQ in our particular set of experiments.

The average learning curves for plays of WoLF against Rmax, presented in Figure 12d, converge to the minimax value within the examined time interval under the following settings: Rmax Attacks and $K1 = 50$, Rmax Defends and $K1 = 50$ or $K1 = 100$. Prior to convergence, WoLF gains return higher than the minimax. In the cases of convergence to minimax value, Rmax also converges to, at least almost, a NE policy. WoLF in these cases does not converge to NE but rather to some other best response (Table 4). The dynamic of the learning process can roughly be divided into stages, defined by the discovery of new entries by Rmax and the following policy updates. Rmax starts off with a uniformly mixed policy. Before new entries become known to Rmax, WoLF learns a deterministic response policy with a higher return to WoLF than the minimax value. In turn, entries associated with WoLF's deterministic policy are the first to become known to Rmax. Each joint policy that results from Rmax's policy updates has one of the following properties: (i) The new joint policy seldom visits unknown entries. This policy provides Rmax with return close to or greater than the minimax. (ii) The new joint policy frequently visits unknown entries and provides Rmax with higher return than its average so far. (iii) The new joint policy frequently visits unknown entries and provides Rmax with return equal or lower than its average return so far. In cases (i) and (ii), WoLF will switch to "learn fast" mode and, unless Rmax is already playing the minimax policy, will manage to learn a new response policy with return higher than the minimax before the next stage. This joint policy is guaranteed to visit unknown entries but directs exploration to entries more profitable to WoLF. WoLF's hill climbing method, variable learning rate and small action space enable it to adjust fast to Rmax's new policies and maintain an average return higher than the minimax until Rmax converges to the NE policy. Since Rmax starts off with highly exploratory policies, WoLF is able to attain high payoffs at an early learning phase and thus maintain a high threshold for determining switching of learning rates. By playing a best response, WoLF directs joint exploration as to delay convergence of Rmax on one hand and encourage fast growth of its estimated Q-values on the other.

The relation between the value of $K1$ in Rmax to the convergence time is not as clear as in self play because higher values for $K1$ give WoLF more time to adapt and exploit each new policy of Rmax. It should be noted that for deterministic models, $K1$ can be set to 1, leading to very rapid convergence of Rmax. However, our parameter selection attempts to optimize for a wide range of environmental dynamics and assumes this dynamic is not known ahead of time.

The phenomena observed in this game were repeated in the games described in the following sections. Therefore, in the following, only phenomena not observed in the 3×3 Wall game are discussed.

4.5.2 5×2 WALL GAME

The transition rules of this game are identical to the 3×3 Wall game. It is played on a 5×2 grid under deterministic transition probabilities. Figure 13 depicts the initial position of the game and A's reward structure. When A reaches the right column of the grid in row i, it is rewarded by r_i, where $r_0 = 100$ and $r_i = r_{i-1} + 10i$ for $i \in \{1,2,3,4\}$, and the agents are reset in their initial positions. Otherwise A is rewarded by 90. The fixed sum of the game is 200. The minimax a.r.p.s. for the Attacker is ~ 105.18. The game is designed to fool GLIELPs with random exploration played by

A. If *A* explores randomly, it will probably discover the high rewards in the top rows before it will discover the higher rewards in the lower rows. Later, its growing bias to exploit will prevent it from sufficiently exploring the lower rows.

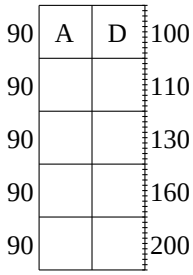
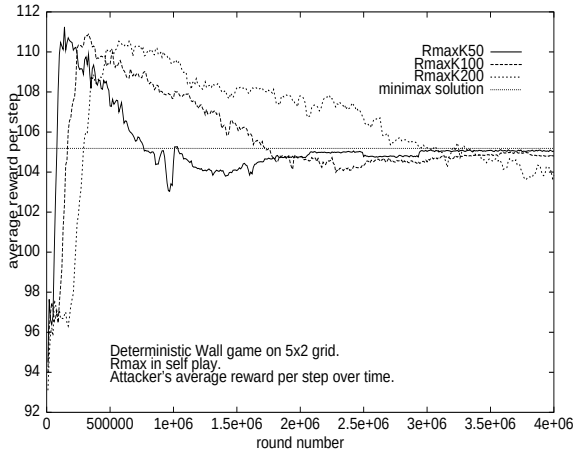
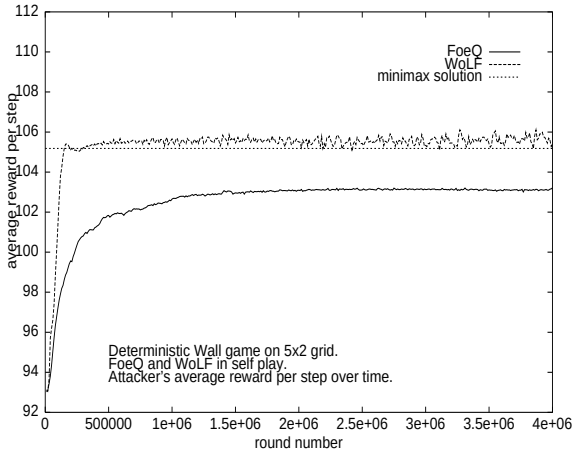


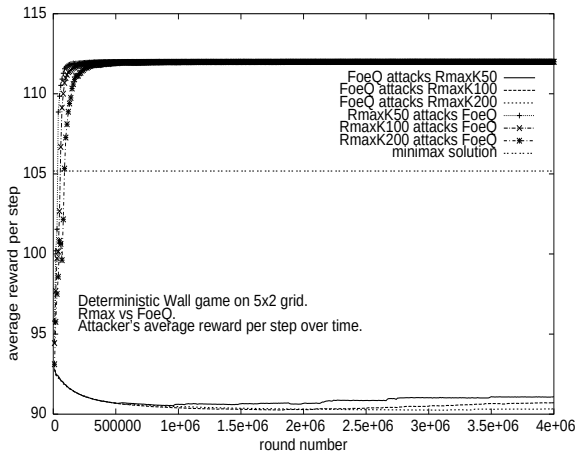
Figure 13: 5×2 wall game – initial position and rewards



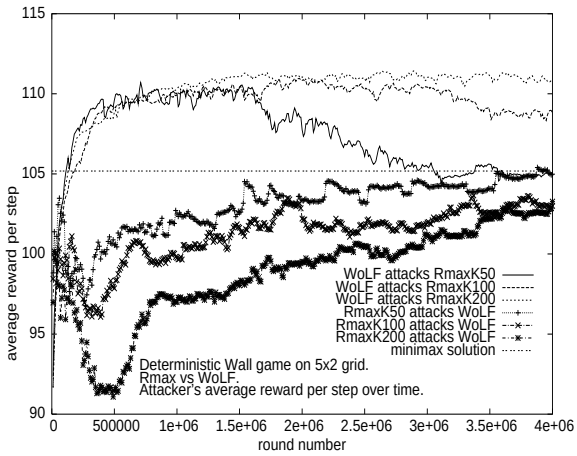
(a) Rmax in self play



(b) FoeQ and WoLF in self play



(c) Rmax vs FoeQ



(d) Rmax vs WoLF

Figure 14: 5×2 wall game – average reward per step

			Attacker							Defender						
A	*D*	*K*1	u	l	d	r	s	Q-val	KE	u	l	d	r	s	Q-val	KE
Opt	**Opt**		**.0**	**.0**	**1.0**	**.0**	**.0**	**5209**	**1125**	**.0**	**.0**	**1.0**	**.0**	**.0**	**4790**	**1125**
RX	RX	50	.0	.0	1.0	.0	.0	—	761	.0	.0	1.0	.0	.0	—	761
RX	RX	100	.0	.0	.971	.029	.0	—	745	.0	.0	1.0	.0	.0	—	745
RX	RX	200	.002	.005	.803	.185	.005	—	723	.0	.051	.903	.046	.0	—	723
FQ	FQ		.648	.0	.0	.306	.045	4783	—	.897	.0	.0	.053	.050	4403	—
WF	WF		.0	.003	.938	.057	.002	5316	—	.001	.0	.999	.0	.0	4742	—
RX	FQ	50	.0	.0	1.0	.0	.0	—	272	.925	.026	.004	.036	.009	507	—
RX	FQ	100	.0	.004	.970	.026	.0	—	229	.949	.006	.0	.023	.022	586	—
RX	FQ	200	.049	.053	.473	.075	.350	—	199	.807	.024	.025	.091	.053	728	—
FQ	RX	50	.213	.081	.158	.293	.255	1726	—	.407	.492	.026	.025	.050	—	88
FQ	RX	100	.252	.193	.014	.319	.222	2861	—	.297	.652	.007	.031	.013	—	58
FQ	RX	200	.312	.165	.033	.318	.172	3926	—	.055	.931	.007	.007	.0	—	43
RX	WF	50	.0	.029	.971	.0	.0	—	456	.0	.0	1.0	.0	.0	4795	—
RX	WF	100	.0	.056	.813	.117	.014	—	438	.0	.0	.992	.004	.004	4832	—
RX	WF	200	.0	.014	.718	.268	.0	—	406	.0	.007	.916	.025	.052	4863	—
WF	RX	50	.0	.0	.996	.004	.0	5223	—	.0	.0	1.0	.0	.0	—	449
WF	RX	100	.025	.049	.856	.021	.050	5400	—	.133	.059	.675	.106	.026	—	361
WF	RX	200	.181	.083	.619	.064	.053	5475	—	.176	.213	.391	.084	.136	—	221

Table 5: 5×2 wall game – NE policies and Average learned policies for state $\langle (2,0),(2,1) \rangle$, average learned Q-values for action $\langle stand, stand \rangle / \langle stand \rangle$ in the initial state by FoeQ/WoLF, respectively, and average number of known entries by Rmax, after 4×10^6 rounds, classified by players' types. See format explanation in Table 4.

Figure 14 presents the a.r.p.s. obtained by the different agents playing the Attacker's role. Table 5 classifies the average learned policies in state $\langle (2,0),(2,1) \rangle$ (both players on middle row), the average learned values for action $\langle stand, stand \rangle / \langle stand \rangle$ in the initial state by the FoeQ/WoLF players and the average number of known entries by the Rmax players, according to the agents playing *A* and *D*. The minimax policy for *A* and *D* when they are both in rows 0, 1 or 2 is to move *down*. When *A* and *D* are both in rows 3 or 4 their minimax policy is mixed.

Self Play

Rmax with $K1 = 50$ converges to the minimax values (Fig. 14a) and policies (Table 5). With values of 100 and 200 of $K1$, Rmax still has a small exploration bias after 4×10^6 rounds (Table 5) and attains almost the minimax value (Fig. 14a). WoLF's convergence to near the minimax value is exceptionally fast in this domain, despite its GLIELP (Fig. 14b). The gradual increase in payoffs for attacking in lower rows both guides exploration and speeds up learning by causing many switches in the learning rate. By employing a high policy learning rate to find a best response, the WoLF Attacker very quickly discovers the high return in the bottom rows and the WoLF Defender follows by defending them. FoeQ on the other hand falls in the designed trap and converges to a suboptimal policy (Fig. 14b). The FoeQ Attacker believes that it can gain higher rewards in the top rows and assigns a high probability to action *up* in state $\langle (2,0),(2,1) \rangle$ instead of *down*. The FoeQ defender, as well, believes that the upper rows are more worth defending and also assigns a high probability to action *up* in this state (Fig. 14b, Table 5).[16]

Heterogeneous Players

Figure 14c shows *A*'s a.r.p.s. over time for plays of FoeQ against Rmax. FoeQ's behavior is

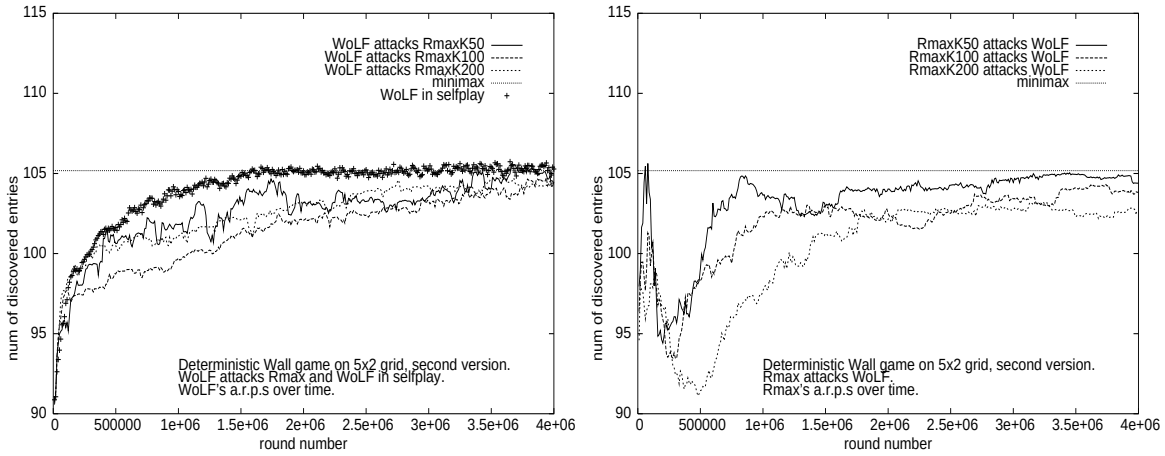
16. Again, this behavior, too, could be attributed to low initial Q-values.

similar to its behavior in self play. When D is played by FoeQ, it attempts to defend the top rows while Rmax attacks in the bottom ones. When A is played by FoeQ, it attempts to Attack only in the top rows (Table 5). Since the entries associated with states of both players being in the bottom rows are unknown to the Rmax Defender, they are modeled as unrewarding by FoeQ, and hence, its policy defends only in the top rows. The low numbers of known entries to Rmax (Table 5) is evidence of FoeQ's inability to sufficiently explore.

In plays of Rmax against WoLF, the players converge to minimax in the examined time interval only for Rmax with $K1 = 50$ and WoLF gains a.r.p.s. higher than the minimax all the way to convergence (Fig. 14d). During the learning period, WoLF's policy is "one step ahead" of Rmax's. In particular, it assigns a greater probability to action *down* in the middle row. This advantage is also expressed by the differences in the probability assigned to action *down* between the learned policies for the different values of $K1$ (Table 5). When WoLF plays the Defender, it adjusts quickly to the behavioral changes of Rmax. When WoLF plays the Attacker, the gradual increase of rewards for attacking in lower rows directs WoLF's exploration while maintaining a high a.r.p.s.

Removing the Intermediate Rewards

To eliminate the incentive to explore and learn quickly given to WoLF by the intermediate rewards, we studied a variant of the 5×2 Wall game: the Attacker's payoffs for attacking in the second and third rows are modified to 100. All the other payoffs and transitions are unchanged. The minimax policies and values are also unchanged since the attacker's optimal policy in both settings attacks only in the two bottom rows.



(a) WoLF attacks Rmax and WoLF in self-play

(b) Rmax attacks WoLF

Figure 15: Modified 5×2 wall game – a.r.p.s. over time

The convergence of WoLF in self play (Fig. 15a) is slower than its convergence on the first reward structure, yet still faster than the other algorithms in self play. In plays of WoLF against Rmax, WoLF has a disadvantage when it plays as the Attacker: The threshold for switching learning rates does not grow in early learning and hence a low learning rate is more frequent while more exploration is required to discover the benefits of attacking in the bottom rows. As an Attacker, WoLF gains a lower return than the minimax value all the way to convergence (Fig. 15a). In the

Defender's role, WoLF's variable learning rate responds to the Rmax Attacker's "initiatives," and WoLF gains higher return than the minimax value all the way to convergence (Fig. 15b).

4.5.3 2×4 TAG GAME

This section describes the results of executing the algorithms on a *stochastic* Tag game. The game is played on a 2×4 grid with a missing corner. One of the players, C, is the tagger and the other, E, is the Escaper. Fig. 16 depicts the initial configuration of the game. A tagging event (tag) occurs when both players have the same positions. In the case of a tag, C receives a reward of 40, E receives a reward of 0 and the players' positions are unchanged. Otherwise C's reward is 15 and E's reward is 25. C's a.r.p.s. under the minimax policy is ~ 18.22. C's optimal strategy is to attempt to trap E in the rightmost cell of the grid while E's optimal strategy is to avoid this situation. To this end, the minimax strategy is deterministic at all states except $\langle (0,2),(0,3) \rangle$.[17] For example, in state $\langle (0,1),(1,1) \rangle$ the Escaper should move *left* and not *right* to avoid the danger of being forced to the corner.

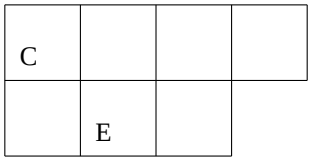


Figure 16: 2x4 tag game – initial position

Figure 17 presents the a.r.p.s. obtained by the different agents playing the tagger's role. Table 6 classifies the average learned policies in state $\langle (0,1),(1,1) \rangle$, the average learned values for action $\langle stand, stand \rangle$ / $\langle stand \rangle$ in state $\langle (0,2),(0,3) \rangle$ by the FoeQ/WoLF players, and the average number of known entries by the Rmax players, after 4×10^6 rounds, according to the agents playing C and E.

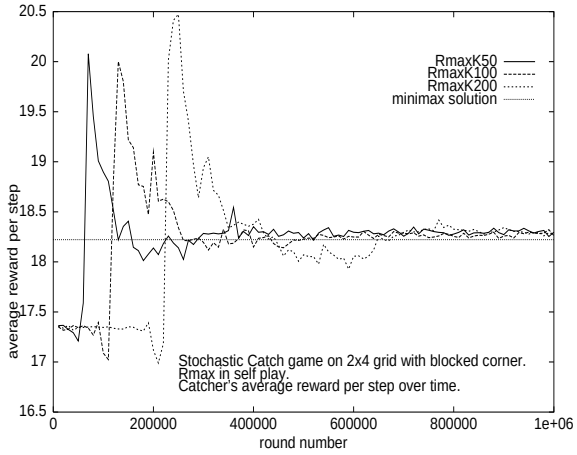
Self Play

In self play, the Rmax Escaper does not learn an optimal policy. Furthermore, in contrast to the deterministic games, E's policy improves with greater values of $K1$, although fewer entries become known (Table 6).[18] This is because more sampling is required to approximate the transition probabilities. However, the learned policies yield an 0.2-optimal return (Fig. 17a), closer to the optimal value than the other algorithms. FoeQ and WoLF converge to near the minimax value within the first 6×10^5 rounds and both learn almost optimal/minimax policies. FoeQ performs much better in this domain than in the previous deterministic domains because the stochastic transitions amplify exploration.

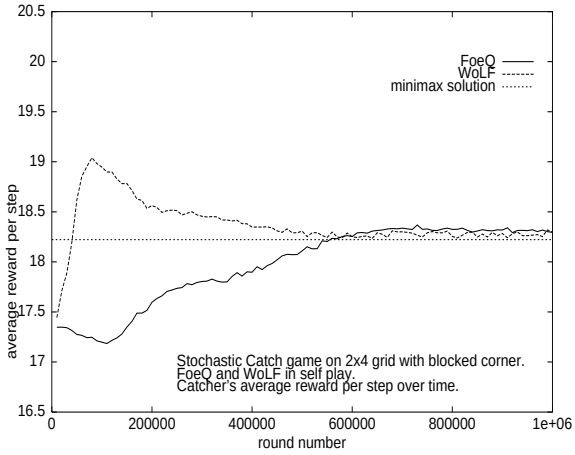
Heterogeneous Players

When Rmax plays against FoeQ, more entries become known and FoeQ's value estimates are better compared to the deterministic games, again due to the amplification of exploration by the

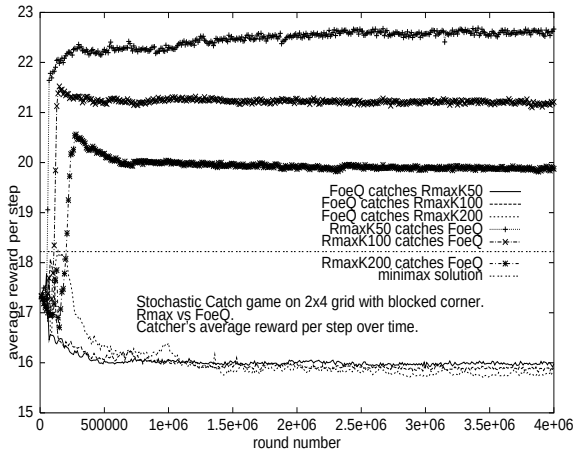
17. Positions are denoted *(row,column)*, with (0,0) being the upper left position. From state $\langle (0,2),(0,3) \rangle$ the players can transit to state $\langle (0,3),(0,2) \rangle$ by the joint action $\langle$*right,left*$\rangle$ without the occurrence of a tag.
18. Recall that as $K1$ increases, more visits are required to mark an entry known. Therefore, fewer entries will be marked within a given time frame.



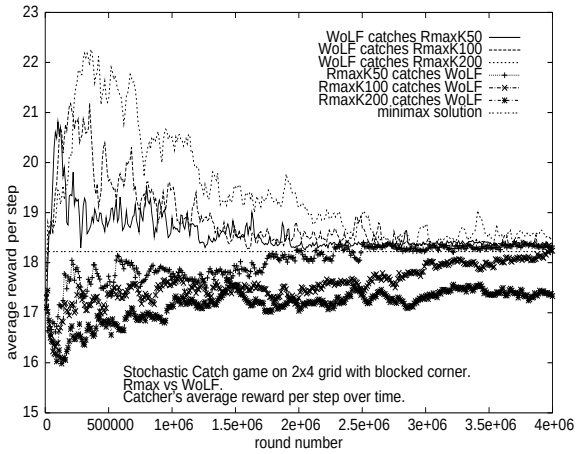
(a) Rmax in self play **(first** 10^6 **rounds)**



(b) FoeQ and WoLF in self play**(first** 10^6 **rounds)**



(c) Rmax vs FoeQ



(d) Rmax vs WoLF

Figure 17: 2×4 tag game – average reward per step

stochastic environmental dynamic. When FoeQ plays Escaper against Rmax, FoeQ learns better policies when Rmax uses larger values of $K1$ (Table 6) and receives a greater average reward (Fig. 17b)—opposite to what was observed in the 3×3 Wall Game. It seems that the longer periods of Rmax playing fixed policies enable FoeQ to better approximate the different Q-values associated with that policy. Despite the stochastic nature of the environment playing "in favor" of FoeQ, Rmax is still superior in heterogeneous play. In plays of Rmax against WoLF, the algorithms converge to the minimax value in five out of the six different configurations, whereas in the deterministic games they converged in two or three out of the six. The convergence dynamic is similar to that observed in the first two games, and WoLF receives an a.r.p.s. greater than the minimax value all the way to convergence.

			Tagger							Escaper						
A	*D*	*K*1	u	l	d	r	s	Q-val	KE	u	l	d	r	s	Q-val	KE
Opt	**Opt**		**.0**	**.0**	**1.0**	**.0**	**.0**	**956**	**1050**	**.0**	**1.0**	**.0**	**.0**	**.0**	**1044**	**1050**
RX	RX	50	.0	.0	1.0	.0	.0	—	844	.0	.450	.0	.550	.0	—	844
RX	RX	100	.0	.0	1.0	.0	.0	—	838	.0	.750	.0	.250	.0	—	838
RX	RX	200	.0	.0	1.0	.0	.0	—	833	.0	.850	.0	.150	.0	—	833
FQ	FQ		.0	.0	.995	.005	.0	889	—	.002	.769	.0	.229	.0	1048	—
WF	WF		.0	.0	1.0	.0	.0	922	—	.005	.655	.0	.340	.0	1089	—
RX	FQ	50	.0	.0	.950	.050	.0	—	742	.138	.278	.164	.225	.195	566	—
RX	FQ	100	.0	.0	1.0	.0	.0	—	649	.067	.150	.064	.618	.101	802	—
RX	FQ	200	.0	.0	1.0	.0	.0	—	572	.038	.320	.061	.541	.040	952	—
FQ	RX	50	.090	.094	.476	.120	.220	448	—	.0	.750	.0	.025	.0	—	789
FQ	RX	100	.112	.029	.471	.049	.339	618	—	.0	.575	.0	.413	.012	—	711
FQ	RX	200	.175	.051	.539	.020	.215	720	—	.0	.461	.001	.438	.100	—	622
RX	WF	50	.0	.0	1.0	.0	.0	—	583	.020	.668	.0	.317	.0	1050	—
RX	WF	100	.0	.0	.983	.0	.017	—	547	.015	.601	.002	.383	.0	1063	—
RX	WF	200	.0	.050	.850	.0	.010	—	488	.020	.392	.001	.531	.056	1098	—
WF	RX	50	.0	.0	.992	.0	.008	973	—	.0	.7	.0	.3	.0	—	608
WF	RX	100	.003	.001	.994	.0	.002	977	—	.0	.65	.0	.35	.0	—	567
WF	RX	200	.032	.019	.928	.008	.013	992	—	.0	0.7	.0	0.3	.0	—	508

Table 6: 2×4 tag game – NE policies and average learned policies for state $\langle (0,1),(1,1) \rangle$, average Q-values for action $\langle stand, stand \rangle$ / $\langle stand \rangle$ in state $\langle (0,2),(0,3) \rangle$ by FoeQ/WoLF and average number of known entries by Rmax, after 4×10^6 rounds, classified by players' types. See format explanation in Table 4.

4.5.4 SUMMARY

The adversarial exploration/exploitation tradeoff in FSSGs is more complicated than that observed in the common interest CISG case. Optimizing behavior during learning introduces a tradeoff between exercising opponents' exploration in order to gain higher return (may-be at the expense of fast convergence to some fixed learning target), to exercising opponents' exploration for joint exploration. When one algorithm takes the time to explore, the other algorithm can exploit and obtain payoff higher than the NE. To this end, learning a best response proves better than learning a NE, when combined in the WoLF algorithm with other properties that ensure fast adaptation to a changing adversary.

Indeed, WoLF appears to be the preferred algorithm in heterogeneous play, with good performance in self-play as well. Nevertheless, WoLF fails to converge to NE in heterogeneous play against an adversary that does converge to a NE, which may be its Achilles heel. WoLF's robustness makes up for the classic weakness of GLIELPs discussed in Section 3.4 (that is, the great sensitivity to the exploration schedule in some domains), but not completely. Thus, while in most cases WoLF is preferable over the other presented algorithms, in some situations this anomaly manifests itself and Rmax outperforms WoLF.

Additional practical issues may affect the choice of algorithm for a specific task. WoLF is computationally more efficient, mainly because it does not involve equilibrium computations. Rmax is much simpler for pre-tuning, with a single intuitive parameter, but requires solving the underlying stochastic game.

5. MGS

MGS is a Markov Game Simulation system designed to evaluate online performance of MARL algorithms. Three main software components take part in a simulation:

Players – user-defined implementations of MARL algorithms.

Referee – a user-defined program that represents a multi-agent environment.

Simulator – mediates between the Players and Referee.

MGS provides Java interfaces and an abstract Referee class that implements the backbone of typical grid-world environments and makes the programming of grid worlds simple and easy. It should be noted that the description of software components and methods in the rest of this section is for illustrative purposes and is partial and incomplete.

Modeling real world environments (or simplifications of such environments) as Stochastic Games is a tedious task for humans. To simplify the modeling task, MGS supports simple creation of grid-world environments referred to as Grid Games (GG). In a GG, agents can move about between squares of a grid, move/carry objects etc. GGs induce MGs in which the set of states S are the possible assignments to the state variables, which are typically the position of the agents and various objects. Actions change the positions of the agents and the state of the objects.

5.1 The Referee

This program represents a GG. The state variables of the GG are memory variables of the Referee program and reachable internal states of the Referee correspond to possible assignments to the state variables. The Referee may manage additional memory variables, that is, variables that capture the previous assignment to the state variables in order to implement the payoff function. The Referee implements methods that simulate the environment such as:

- `getStateIndex()` - enumerates the state space. Returns a unique integer that corresponds to the current state of the Referee.
- `giveActions(int[] actions)` - receives the action choices of the players and updates the state variables to characterize the new state of the environment.
- `getPlayerReward(int p)` - returns the payoff for player `p`.

5.2 The Players

They implement the methods:

- `play(int s)` - returns the action choice in state `s`.
- `update(int s, int[] acts, double r)` - updates the algorithm's model / values according to the new state `s`, other Players' actions `acts` and payoff `r`.

5.3 The Simulator

This module is the active process during simulation. Schematically, the Simulator loops over the following steps:

1. get the Players' actions in the current state.
2. pass the joint action to the Referee.
3. compute the index of the new state of the environment.
4. pass the new state index and payoffs to the players.

Typically, GGs involve actions that move the agents up, down, left and right on a two-dimensional grid. To unburden the user from modeling these aspects of the environment, they are already built into the system. The abstract Referee class implements various methods for manipulating the positions of the agents on a grid represented by a two dimensional integer array. The Simulator also computes the new positions of the agents according to the five default actions `up`, `left`, `down`, `right` and `stand`, and according to user input transition probabilities. In Step 2 above, the Simulator also passes to the Referee the results of this computation in the form of suggested new positions for the players.

MGS is a very flexible tool. Despite the implemented GG features, it can in fact be used to model *any* discrete state-action space MG (although doing so may require more complicated programming than the simple implementation of GGs). MGS offers various features that make it a convenient experimental tool. Input can be specified either by a GUI or by a script. Scripts may specify multiple independent simulations and may also include parameters for the Players' algorithms. MGS logs statistics of payoffs and selected actions and also supports logging by the Players. For further information on MGS, see `http://www.cs.bgu.ac.il/~mal`.

6. Conclusions

This paper presents a large empirical study of representative MARL algorithms conducted using the MGS tool. Such comprehensive studies in this area are rare. The only other related study we are aware of appeared in Powers and Shoham (2005) and involved the much simpler class of repeated games with known game matrices. While most authors run some empirical studies, these often focus on their algorithms and do provide a comprehensive comparison of strengths and weaknesses.

We believe that our results and analysis can serve to guide researchers in developing more powerful algorithms and formal analytic tools, and practitioners in selecting and tuning algorithms for specific tasks. Some of our results are closely related to phenomena observed in single-agent reinforcement-learning algorithms, especially in common-interest environments, which can be viewed as describing a distributed version of single-agent RL. In this domain, the issue of exploration vs. exploitation appears to play a major role in the success of different algorithms. Here, we found Rmax's exhaustive approach to be very useful, being much less susceptible to being stuck in local minima compared to GLIELP exploration. Of course, one does expect this almost-exhaustive exploration approach to be costly in large domains. However, in our examples, Rmax was able to perform well with small sample sizes, partly due to the locality of actions—that is, the fact that most actions have a small number of possible outcomes and these effects do not change the agent's state drastically. In general, many real-world domains tend to have this property. We believe that

online identification of locality properties may be used to construct more practical variants of Rmax as well as other methods. Rmax also provides another capability we found important in common interest games: coordinated exploration. It also seems to scale well with the number of agents. Perhaps most important is the fact that it is very simple to understand its behavior, and consequently, we believe, to modify it given background knowledge. However, Rmax is completely inadequate if cooperation is to be obtained given a system with heterogeneous agents.

Fixed sum games provided an interesting setting, where we could test algorithms against each other. We found that learning is more efficient when the greedy component of the learning policy is a best response rather than a minimax strategy. The WoLF algorithm achieves fast adaptation to a changing opponent by maintaining values for only the private action space and by regulating behavior according to the dynamics of the learning process, and it seems to be the best choice in such competitive environments.

Overall, it seems that there is much potential for improved performance by multi-agent learning algorithms. We hope this study will motivate the design of algorithms that improve upon the current state of the art, and we believe that the MGS test bed can be a useful tool for testing such new techniques.

Acknowledgments

We are grateful to the reviewers for their useful suggestions and comments, and to the associate editor, Michael Littman, for his many useful and detailed comments. Partial support was provided by the Paul Ivanier Center for Robotics Research and Production Management, by the Lynn and William Frankel Center for Computer Science, and by the Israel Science Foundation.

References

A. Bab and R. I. Brafman. An experimental study of different approaches to reinforcement learning in common interest stochastic games. In *ECML*, pages 75–86, 2004.

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

R. I. Brafman and M. Tennenholtz. R-max – a general polynomial time algorithm for near-optimal reinforcement learning. *JMLR*, 3:213–231, 2002.

R. I. Brafman and M. Tennenholtz. Learning to coordinate efficiently: A model based approach. *JAIR*, 19:11–23, 2003.

R I. Brafman and M. Tennenholtz. Efficient learning equilibrium. *Artif. Intell.*, 159:27–47, 2004.

G. W. Brown. Iterative solution of games by fictitious play. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*. Wiley, 1951.

G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning: A Bayesian approach. In *AAMAS'03*, 2003.

C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multi-agent systems. In *Proc. Workshop on Multi-Agent Learning*, 1997.

R. Dearden, N. Friedman, and D. Andre. Model based bayesian exploration. In *UAI'99*, 1999.

R. Dearden, N. Friedman, and S. Russell. Bayesian Q-learning. In *AAAI-98*, 1998.

E. Even-Dar and Y. Mansour. Learning rates for *Q*-learning. *Journal of Machine Learning Research*, 5:1–25, 2003.

J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.

J. Hu and M.P. Wellman. Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML-98)*, pages 1095–1100, 1998.

L. P. Kaelbling. *Learning in Embedded Systems*. The MIT Press: Cambridge, MA, 1993.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

M. L. Littman. Friend-or-foe Q-learning in general-sum games. In *Proc. 18th International Conf. on Machine Learning*, 2001.

M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. 11th International Conference on Machine Learning*, 1994.

A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

R. Powers and Y. Shoham. Learning against opponents with bounded memory. In *Proc. 19th International Joint Conf. on Artificial Intelligence*, 2005.

M. Puterman. *Markov Decision Processes*. Wiley, New York, 1994.

A. Schaerf, Y. Shoham, and M. Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500, 1995.

Y. Shoham, R. Powers, and T. Grenager. If Multi-Agent Learning is the Answer, What is the Question? *Artificial Intelligence*, 171(7):365–377, 2007.

S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3):123–158, 1996.

M. Sridharan and G. Tesauro. Multi-agent Q-learning and regression trees for automated pricing decisions. In *Proc. 17th International Conf. on Machine Learning*, pages 927–934. Morgan Kaufmann, San Francisco, CA, 2000.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

W. Uther and M. Veloso. Adversarial reinforcement learning. Technical report, Carnegie Mellon University, 2003.

J. M. Vidal and E. H. Durfee. Predicting the expected behavior of agents that learn about agents: the clri framework. *Autonomous Agents and Multi-Agent Systems*, 6(1):77–107, 2003.

R. V. Vohraa and M. P. Wellman. Foundations of multi-agent learning: Introduction to the special issue. *Artificial Intelligence*, 7:363–364, 2007.

X. Wang and T. Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In *NIPS'02*, 2002.

Z. Zheng, M. Shu-gen, C. Bing-gang, Z. Li-ping, and L. Bin. Multiagent reinforcement learning for a planetary exploration multirobot system. In *PRIMA*, pages 339–350, 2006.