Learning to cooperate in multi-agent systems by combining Q-learning and evolutionary strategy

Mary McGlohon and Sandip Sen
mary-mcglohon@utulsa.edu, sandip@utulsa.edu
Department of Computer Science
The University of Tulsa, Tulsa, Oklahoma

Abstract

Coordination games can represent interactions between multiple agents in many real-life situations. Thus single-stage coordination games provide a stylized, abstracted environment for testing algorithms that allow artificial agents to learn to cooperate in such settings. Individual reinforcement learners often fail to learn coordinated behavior. Using an evolutionary approach to strategy selection can produce optimal joint behavior but may require significant computational effort. Our goal in this paper is to improve convergence to optimal behavior with reduced computational effort by combining learning and evolutionary techniques. In particular, we show that by letting agents learn in between generations in an evolutionary algorithm allows them to more consistently learn effective cooperative behavior even in difficult, stochastic environments. Our combined mechanism is a novel improvisation involving selecting actual rather than inherited behaviors.

1 Introduction

Reinforcement learning and evolutionary computation are active topics in the area of autonomous agents due both to their generic applicability and to the aesthetic appeal of their similarities to biological systems. On the other hand, coordination in games has applications in the behavioral and social sciences. In this paper, we use instances of machine learning and adaptation techniques, Q-learning and evolutionary strategy, to learn to solve cooperative games.

Many games are modeled as matrices representing different action choices available to players. In coordination games, actions along the diagonal of the matrix have higher payoffs than the other actions. Our work uses a special kind of coordination game, that which gives equivalent payoffs to both players given an action combination. When the games are repeated, it is beneficial for an agent to learn which of its own action choices will produce the highest payoff, given the opponent's policy.

Traditional, single-agent machine learning algorithms applied to cooperative systems are not guarateed to converge to the optimal action combination. Without sufficient exploration, an agent can get "stuck in a rut" and choose nonoptimal actions because the payoff seems good enough. Also, if the domain is nondeterministic, an agent may get discouraged with infrequent, but significantly low payoffs for an action, even when choosing that action is optimal [3, 5]. Therefore, using an algorithm with a proper balance of exploration and exploitation is desirable.

In our work, we have designed a combined evolutionary and reinforcement learning approach to solve some particular coordination games. A population of agents plays a game for several iterations, and the most successful agents pass phenotypic information on to the offspring. The offspring then improve on the parents' policy through a reinforcement learning algorithm. This will often cause later generations to converge to optimal behavior in the coordination game.

2 Related Work

2.1 Single-stage coordination games

Single-stage coordination games have been used as a model for studying coordination in multi-agent systems [3, 5]. A *stage game* consists of a set α of n agents, where each agent $i \in \alpha$ has a set A_i of *individual actions*. The agents simulateously choose actions from their respective A_is each time they play the stage game. A set of payoff matrices, R_i, specify the payoff to each player for each possible action combination. In coordination games payoffs are highest along the diagonal; in the games in this work, all players will also receive the same payoff for each action combination. We use π^i to denote the strategy profile for agent i, where $\pi^i(a)$ is the probability of choosing action $a \in A_i$ by the agent i in repeated play of stage games. $\prod$ denotes the product of the individual strategies. If each π_i is deterministic, i.e., $\pi^i(a) = 1$ for some $a \in A_i$, then $\prod$ is a *joint action* [3, 5]. Multiagent learning schemes attempt to converge to coordinated actions.

The single-stage coordination games used in this paper

	$1a$	$1b$	$1c$
$2a$	11	-30	0
$2b$	-30	7	6
$2c$	0	0	5

Table 1: Payoff matrix for the climbing game.

	$1a$	$1b$	$1c$
$2a$	10	0	k
$2b$	0	2	0
$2c$	k	0	10

Table 2: Payoff matrix for the penalty game.

	$1a$	$1b$	$1c$
$2a$	11	-30	0
$2b$	-30	14/0	6
$2c$	0	0	5

Table 3: Payoff matrix for the partially stochastic game.

	$1a$	$1b$	$1c$
$2a$	10/12	5/-65	8/-8
$2b$	5/-65	14/0	12/0
$2c$	5/-5	5/-5	10/0

Table 4: Payoff matrix for the fully stochastic game.

include two deterministic games: the climbing game and the penalty game [3]. The reward matrices for the players for these two games are shown in Tables 1 and 2. These games are interesting because the optimal action choice is different from the safest action choice, the action with the maximum worst-case payoff. For instance, in the climbing game, the optimal action combination is "a,a", but if one of the two agents chooses "b" instead of "a," both agents are penalized heavily. Action "c" is the safest action: no matter what the other agent chooses, the payoff is nonnegative. Therefore, the action combination agents choose may converge to "c,c" and result in a suboptimal payoff. If they learn to cooperate, however, they may learn to choose the optimal action and thus improve their payoffs. The penalty matrix has a similar dilemma: action "b" is the safest option, but if "a,c" or "c,a" combinations are chosen, agents are penalized by value k. To make the games more challenging, one can vary the climbing game to make it partially or fully stochastic, as shown in Tables 3 and 4 [5]. This means that the same action combination will return different payoffs with some static probability distribution. For the partially stochastic game, the action combination "b,b" will pay 0 or 14– each payoff being equally likely. All other action combinations give payoffs as in the climbing game. In the fully stochastic game, each action combination will give two equally likely payoffs. In both the stochastic games, the payoffs average to the same values as in the deterministic climbing game.

2.2 Reinforcement learning

Reinforcement learning [4] is a general feedback-based method for machine learning. It allows an agent to make decisions and predictions for the optimal action without requiring a teacher or supervisor to specify the best action choice. Instead of taking advice, an agent receives *rewards* or *reinforcements* as a result of its actions [10]. The reward, typically a scalar value, represents the outcome of an interaction between the agent and its environment [11]. The simplest agent, a memoryless *reflex agent*, learns to directly map states to actions. A *utility-based agent* learns the utility function of states and uses it to decide actions. Finally, a *Q-learning* agent learns a policy function and uses the function to map states to actions that will maximize reward [10].

Q-learning, developed by Watkins [12], is the most frequenty used reinforcement learning algorithm. An agent estimates the utility for doing each of its actions, chooses an action based on a selection function of the expected values, observes a reward, and then updates the Q-value or the estimate of the utility of that action. The Q-update function in stateless games is:

$$Q(a) \leftarrow Q(a) + \lambda(r - Q(a)) \tag{1}$$

where r is the reward received, and λ is the learning rate ($0 < \lambda < 1$).

The action selection function is important as effective learning requires sufficient exploration. Researchers often use a Boltzmann distribution [3, 4, 5] for the probability of choosing an action:

$$\pi(a^i) = \frac{e^{\frac{EV(a^i)}{T}}}{\sum_{a^{i\prime} \in A_i} e^{\frac{EV(a^{i\prime})}{T}}} \tag{2}$$

where EV is the expected value of an action (conventionally the Q-value itself) and T is the temperature. Temperature determines the likelihood for an agent to explore other actions: for high temperature, even when an expected value of a given action is high, an agent may still choose an action that appears less desirable. This exploration is especially necessary in stochastic games, where payoffs received for the same action combination may vary. For effective exploration, a high temperature is used in the early stage games. The temperature is decreased over time to favor more exploitation, as the agent is more likely to have discovered the true estimates of different actions. The temperature as a function of iterations is given by:

$$T(x) = (e^{-sx} * T_{max}) + 1 \tag{3}$$

where x is the iteration number, s is the rate of decay and T_{max} is the starting temperature,

Claus and Boutilier present two different forms of multiagent reinforcement learning. *Independent learners* act blind to other agents in play; only the individual action a^i, not the joint action, is taken into account when updating action utility estimates upon receiving the reward. *Joint action learners* (JALs), on the other hand, consider the actions of the other agents in the system. Whereas independent learners only keep track of Q-values for each A_i in the action space, joint action learners keep a Q-table that includes separate entries for each set of possible joint actions, giving the Q-table $|A|^n$ entries, where $|A|$ denotes the number of possible actions, and n is the number of learners.

The JAL algorithm takes the probability of a *joint* action being taken and factors it into the Q-value. Given an experience $\langle a_i, o, r\rangle$, where a^i is the action taken by agent i and o the observation resulting from the action, there are different possible joint actions a that support the situation. Thus, the Q-update function is:

$$Q(a) \leftarrow Q(a) + \lambda Pr(a|o, a^i)(r - Q(a)) \tag{4}$$

where $Pr(a|o, a^i)$ is the probability that action a and outcome o both occur (the joint action experience observation), given an action a^i.

Claus and Boutilier showed that JALs will converge faster than ILs will. This is similar to *fictitious play*, in which agents play best response to the observed opponent [8]. When agents explicitly reason about other agents, as in JAL, it can facilitate convergence in coordinated games [2]. However, JALs are inefficient in more complicated games with more agents, as the joint action space increases exponentially with increasing n. Therefore, it is helpful to find a learning algortihm for ILs that converges to desireable outcomes for such games. It is necessary to consider that Q-learning for ILs guarantees convergence only in single-agent settings [3]; when it does converge it will often converge to sub-optimal behavior. To confront this problem, Kapetanakis and Kudenko explored different reinforcement learning heuristics in the deterministic and stochastic games– some heuristics performed better on deterministic games than did straightforward Q-learning; however, a solution was not found for the fully stochastic game at that time [5]. In more recent work, they found a solution to the fully stochastic domain, by using commitment sequences [6]. Such commitment sequences, however, require pre-play "locker-room agreements," which may not always be feasible.

Another way to teach cooperation in a multi-agent system is to use a *joint policy table* [7]. In it, agents in a system each contribute to the Q-table. The drawback of this system is that it introduces an element of *common knowledge*, and in some way defeats the purpose of a multiagent system; instead of having individual decision processes, the agents are to some extent different parts to one controlling superagent [8].

2.3 Evolutionary computation and genetic algorithms

A genetic algorithm, first developed by John Holland, is a method of optimizing a function, based on the theory of evolution [13]. An agent in a genetic algorithm contains a genetic code, where *genes* contain data that influence how an agent performs during its lifetime. After a given "lifetime," the population goes through a *selection* process where all agents are evaluated according to an evaluation function, and then the best are "selected" to reproduce, whether through direct cloning or *crossover*, where each agent in the next generation contains a mixture of the genes of multiple agents in the previous generation. *Mutation* adds to the reproduction of agents a random element. Very infrequently, instead of a parent's gene, a random gene is inserted into the offspring [13].

Ackley and Littman showed that reinforcement learning combined with a genetic algorithm, an approach which they dubbed evolutionary reinforcement learning (ERL) performed better than either strategy alone, in an *artificial life world*, a digital ecosystem [1]. They explained the *Baldwin effect*, with which learned behaviors of agents in a genetic algorithm become part of the genetic code, and agents are born with an instinct to perform certain actions.

3 PAES and PAES-Q

We now present our Parental Advisory Evolutionary Strategy (PAES) system, a modification of the classical genetic algorithm. The population in PAES consists of structures where each structure represents a policy for an agent, where a policy π is a probabilistic distribution over the actions available to an agent. For instance, $\pi = \{0.4, 0.3, 0.3\}$ indicates an agent has a 40 percent chance of choosing action "a", and a 30 percent chance each for choosing "b" or "c." Agents are paired at random from the population and are evaluated by repeatedly playing the game where agent actions are chosen based on their corresponding policies.

Each agent keeps track of the frequency of its actions chosen. Then, agents are chosen to reproduce, based on a fitness function. The probability P of an agent being chosen to reproduce is

$$P(i) = \frac{TotalReward(i)}{\sum_{i' \in \alpha} TotalReward(i')} \tag{5}$$

There is no crossover in our model. Instead, the parent's frequency of actions becomes the child's *probability* of

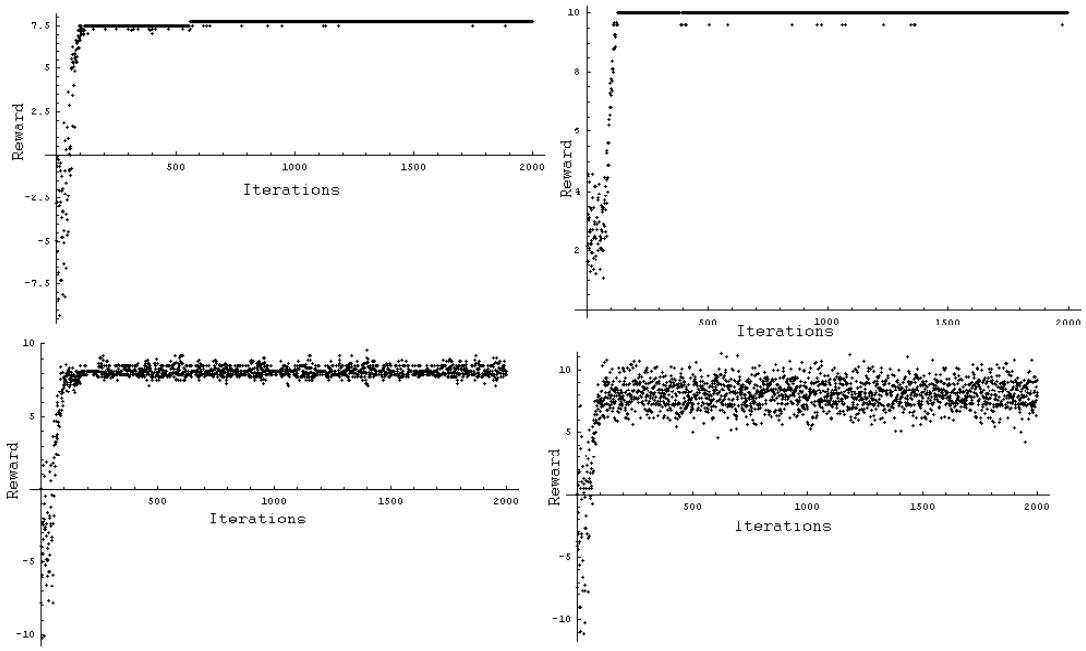

Figure 1: Q-learning: The climbing (top left), penalty (top right), partially stochastic (bottom left), and fully stochastic (bottom right) games, averaged over 10 runs.

choosing an action:

$$\pi_{parent}(a) = \frac{freq_{parent}(a)}{\sum a' \in A_i freq_{parent}(a')} \quad (6)$$

While agents are selected for reproduction proportional to their performance, the offspring policy follows not the policy of the parent, but the actual action frequencies that were sampled from those policies. This copy process rewards or selects actual behavior followed, which can differ (due to sampling effects) from the genetic prescription in the parent policies. Such propagation of actual behavior rather than exact genetic blueprint differs from the copying of the genetic material used in traditional evolutionary techniques. It also supports phenomena similar to the Baldwin effect.

We used a mutation rate of 0.01: for approximately 1 in 100 agents, instead of inheriting the parent's probabilistic distribution over actions, the agent was given three random values summing to 1. The learning algorithm would then proceed as if the random values had been inherited from the parent.

We developed a PAES-Q learner by incorporating Q-learning into this evolutionary algorithm. Instead of agents blindly accepting "parental advice" as the probability of choosing an action, PAES-Q learners can use Q-learning to improve on the behavior inherited from parents. The Q-table setup is as described above, but the action selection algorithm is modified as follows:

$$\pi_{child}(a) = \frac{\pi_{parent}(a) * e^{\frac{EV(a)}{T}}}{\sum_{a' \in A_i} \pi_{parent}(a') * e^{\frac{EV(a)}{T}}} \quad (7)$$

4 Experimental Results

In this paper we consider concurrent learning by two agents, i.e., $n = 2$. The Q-update function used was that in Equation 1, with $\lambda = 0.9$, $T_{max} = 500$, and $s = 0.06$. We bound the lower limit of temperature to be 1 to ensure baseline residual exploration, an set at 0.06 and 500 in the experiments of Kapetanakis and Kudenko, but can be adjusted according to the number of iterations in the experiment [5].

4.1 Results from the climbing and penalty games

We used the games described above to test the algorithms to compare the convergence of classical Q-learning, PAES, and a combination of the two, PAES-Q.

The evolutionary algorithm set up was a population of 200 agents over 200 generations. In one generation, each agent played a single-stage game 1000 times with another agent in the population. In each game, an agent uses a static probability distribution for choosing actions. The starting population was initiated with random probabilities.

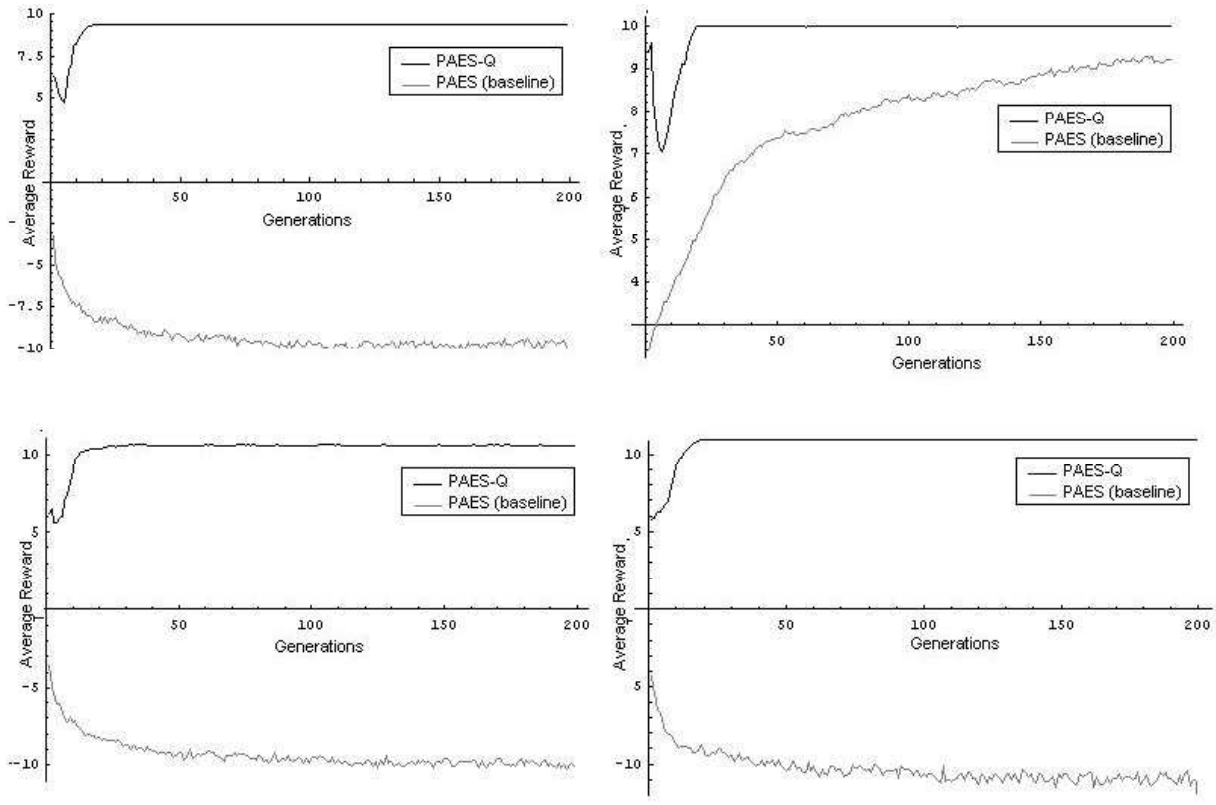

Figure 2: PAES vs. PAES with Q-learning: The climbing (top left), penalty (top right), partially stochastic (bottom left), and fully stochastic (bottom right) games, averaged over 10 runs.

Our results verified previous research that simple Q-learning will converge to nonoptimal choices in all four of the games presented. Repeating this procedure without passing on "parental advice" will inevitably fall short of the optimal action. Results of classical Q-learning in our experiments, which coincide with results of Kapetanakis and Kudenko[5], are shown in Figure 1. Notice that, in the climbing game, the agents are converging to an action with reward of 7, which is the action combination "b,b". The optimal action combination, "a,a" has a reward of 11. The penalty game does indeed converge to the optimal action, where $k = 0$. (Repeated experiments showed that when the penalty is increased, convergence to optimal actions becomes less and less frequent– as action combinations "a,c" and "c,a" return more negative rewards, agents will more commonly converge to action combination "b,b". Again, in stochastic domains, action choice converges to suboptimal average reward.

Similarly, using baseline PAES assured that the agents in the population would converge to optimal outcome after enough generations, but only in the penalty game. In the climbing game, and the stochastic variations thereof, PAES agents fare even worse than Q-learners. However, when the agents in the population also incorporated Q-learning in the action decision process, the population adapted to choose the optimal action (and did so more quickly than PAES in the case of the penalty game). Comparisons between PAES and PAES-Q are shown in Figure 2.

4.2 Results from the diagonal game

To further test PAES-Q, we used a diagonally-optimal matrix. It differs from the others in that all actions are equally beneficial: regardless of the action pair chosen, payoff is 1 if the agents' actions are identical and 0 otherwise.

	$1a$	$1b$	$1c$
$2a$	1	0	0
$2b$	0	1	0
$2c$	0	0	1

Table 5: Payoff matrix for the diagonal game.

Results for this matrix were surprising: despite the multiple options for agents, the population still converged to cooperation. Even though the initial population was randomly distributed across the actions, the fitness function could detect which agents were performing well. Through

the generations, agents that were cooperating with the "popular" action were selected, and the "popular" action choice was passed on to the next generation. PAES-Q converged much more quickly than did PAES alone. In multiple runs, the agents converged to different "popular" actions. Figure 3 compares the performance of PAES to that of PAES-Q. Obviously, since the first generation was not optimal, Q-learning would not have converged by itself. Figure 4 is an example of the distribution of action choices over generations in one run: it shows how action "a", in this case, pulls out over the others.

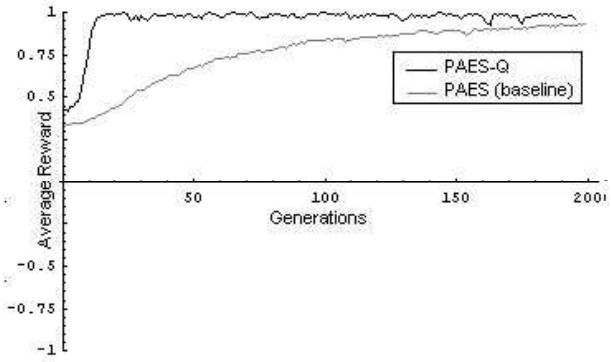

Figure 3: PAES vs. PAES with Q-learning: Diagonal game

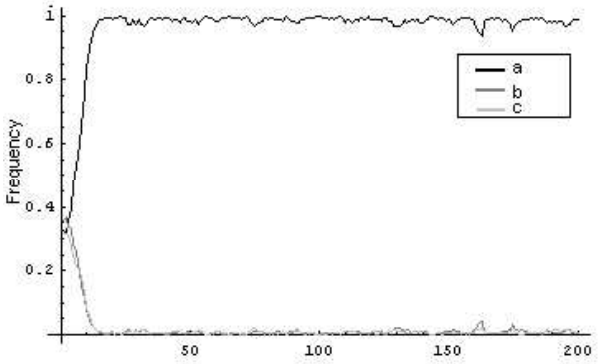

Figure 4: Distribution of actions in PAES-Q over generations: Diagonal game.

5 Future Work

Problems arose when PAES was applied to a game with payoff matrix as shown in Table 6 where optimal joint action involves non-identical choices; that is, when applied to an identical-payoff game that was not a coordination game. When both players had the same optimal action, the method worked, but if one player's highest-paying action was "a" only when another chose "b", or if choosing "b" was optimal only when the opponent chose "c", the agents were unable to converge to an optimal policy. Agents would choose "b" or "c" in the final generation, but would not always produce "b,c" action combinations. In these games, it might be necessary to use cooperative evolution techniques, with two populations: one for Player 1 and one for Player 2. Such work is described in [9]. We plan to incoporate PAES-Q into such a cooperative coevolution framework to solve non-cooperative matrices.

	$1a$	$1b$	$1c$
$2a$	10	20	0
$2b$	20	5	20
$2c$	0	20	10

Table 6: Payoff matrix for an anti-coordination game.

6 Conclusions

Evolutionary strategies using the process of natural selection can be used to learn optimal action combinations in repeated coordination games. We devised an PAES, an evolutionary strategy where offspring adapt parental behavior. The computation involved in an evolving population is significantly greater than that of Q-learning alone. However, Q-learning is not guaranteed to converge to optimal behavior in such multiagent learning scenarios. We proposed PAES-Q, where offspring used Q-learning to improve on behaviors inherited through genetic operators from parents. In many cases, combining Q-learning with PAES causes better results than either strategy used alone. Self-play, however, does not suffice in mixed-strategy games where agents need to follow distinct policies.

Acknowledgments: This work has been supported in part by an NSF award IIS-0209208.

References

[1] David H. Ackley and Michael L. Littman. *Interactions between Learning and Evolution*, pages 487–509. Addison, 1992.

[2] E. Alonso, M. D'Iverno, D. Kudenko, M. Luck, and J. Noble. Learning in agents and multi-agent systems. *Knowledge Engineering Review*, 16:277–284, 2001.

[3] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 746–752. American Association for Artificial Intelligence, 1998.

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

[5] Spiros Kapetanakis and Daniel Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *AAAI/IAAI 2002*, pages 326–331, 2002.

[6] Spiros Kapetanakis and Daniel Kudenko. Reinforcement learning to coordinate using commitment sequences in cooperative multi-agent systems. In *AAMAS03*, 2003.

[7] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. Technical Report GMU-CS-TR-2003-1, Department of Computer Science, George Mason University, 4400 University Drive MS 4A5, Fairfax, VA 22030-4444 USA, 2003.

[8] David C. Parkes and Lyle H. Ungar. Learning and adaption in multiagent systems,. In *AAAI-97*, July 30, 1997.

[9] Narendra Puppala, Sandip Sen, and Maria Gordin. Shared memory based cooperative coevolution. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 570–574, Anchorage, Alaska, USA, 5-9 May 1998. IEEE Press.

[10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT press, Cambridge,MA, 1998.

[12] C. J. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge university, 1989.

[13] Darrell Whitley. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831, 2001.