Reinforcement Learning Algorithms for Homogenous Multi-Agent Systems

R. Matthew Kretchmar
Department of Mathematics and Computer Science
Denison University
Granville, OH 43023
kretchmar@denison.edu

Abstract

We examine the situation in which multiple reinforcement learning agents attempt to speed up the learning process by sharing information. The agents do not interact with each other in the enviroment; instead each agent is learning the same task in isolation from the other agents. As reinforcement learning is a trial-and-error approach, an agent's performance in the environment typically increases with the level of experience an agent has in interacting with that environment. We show how the agents can share Q-values in order to arrive at an optimal policy using fewer trials.

1 Introduction

Many reinforcement learning tasks require extensive learning experience in order to achieve sound performance. Multi-agent systems (MAS) are able to increase the speed of learning by communicating/sharing information during the learning process. While there are a multitude of different MAS learning algorithms, we concentrate on an approach that will be amenable to a parallel hardware application in a future study. Specifically we assume that all the agents are identical, they are all interacting with the same environment in order to learn the same task, and they are all independent of each other; agents do not interact with each other within the environment.

There is a recently active history of research and publication in MAS learning. Gerhard Weiss provides a survey of work in this field and identifies the key issues of learning in MAS [13]. Tan is among the first to publish in this area [11]. In his work, he identifies three key ways MAS can benefit from cooperation: sharing perceptions, sharing raw environment interactions (state-action-reward-state values), and sharing learned policies. Nunes studies an environment of heterogeneous learning agents in which each provides advice that is then incorporated via supervised learning [7]. Littman studies competing RL agents within the context of Markov games [4, 5]. There are many published papers in which agents cooperate to solve different parts of a task [9, 2, 6]. In Bagnell [1], multiple RL robots learn in parallel by broadcasting learning tuples in real time. However in Bagnell's work, parallel RL is only used as a means to study other behavior; parallel RL is not the object of investigation. A parallel Q-learning algorithm is used by Laurent and Piat to solve parts of a block-pushing problem [3]; agents combine their experi-

ence in a globally accessible Q-value table. Printista, Errecalde and Montoya provide a parallel implementation of Q-learning in which each agent learns part of the Q-values and then shares these with other agents [8].

Our study has the following characteristics. The agents are all learning the same environment and are internally identical (same agent architecture); hence we refer to this type of MAS as "homogenous". Importantly the agents are all isolated and independent; they do not interact with each other within the environment. Our method of communication is to share Q-values at different points during the learning process. Our eventual goal is to implement the Q-learning algorithm on parallel hardware. Though we do not address any parallel implementations here, this type of homogenous MAS must first be studied to see which type of information exchange in viable in the MAS environment.

In Section 2 we review the basics of the reinforcement learning. Those familiar with this learning method may skip or browse this section. Section 3 discusses many of the issues involved in transforming the basic reinforcement learning algorithm for the multi-agent environment. We propose two MAS learning algorithms and sketch the operation of each. An example task is presented in Section 4 along with experimental results for our two sharing algorithms. Finally we present some concluding remarks and comment about on-going work in this area in Section 5.

2 Reinforcement Learning

Reinforcement Learning (RL) is the algorithmic embodiment of human learning by trial-and-error. Simply, an agent interacts with an environment by trying various actions to ascertain those that yield the best average rewards. Unlike a supervised learning agent that has an instructor retrospectively providing the optimal action choice for each situation, the RL agent must discover these optimal actions on their own by trying all the actions and selecting the best one. Sutton and Barto provide an excellent text on RL [10]. We utilize the basic Q-learning algorithm first proposed by Watkins [12].

2.1 RL Overview

Formally, the environment is modeled by a Markov decision process. The agent perceives its current state, s_i and then must select an action a from the set of available actions. As a result of the action selection, the agent moves to a new state s_j and receives a reward signal r. In episodic tasks, the agent continues to interact with the environment until a goal or halting state is reached. The agent's objective is to select the set of actions, one per state, that will achieve the maximum sum of reward signals.

A few considerations complicate the learning process. First, there is often a trade-off between short-term and long-term gains; the agent may need to forego a large immediate reward in order to reach states that yield even better possibilities. Another consideration is premature convergence to a suboptimal set of actions. As the agent continues to interact with the environment it must decide between selecting an action that it currently believes is optimal vs selecting another action in the quest for an even better action; this situation of *exploitation* vs *exploration* arises because the environment is often stochastic.

2.2 Algorithms

Temporal difference learning is a hybrid approach that combines the best of dynamic programming and Monte Carlo methods. Q-learning is the most common and well-studied variant of temporal difference learning [12]. Essentially a table of Q-values is maintained with an entry for each state/action pair. A Q-value, $Q(s, a)$, is an estimate of the expected sum of future rewards that the agent is likely to encounter when starting in state s and initially selecting action a; this sum includes not only the immediate reward signal but also all the other rewards accumulated on the way to the goal state. The purpose of reinforcement learning is to discover these Q-values empirically. If the agent has a complete table, then the agent may interact with the environment optimally by searching through the set of available actions for the current state and selecting the table entry $Q(s, a)$ with the maximum value.

The basic update equation for Q-learning is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \cdot TDerror, \quad (1)$$

$$TDerror = \max_{a'} \{Q(s', a')\} + r - Q(s, a), \quad (2)$$

where $\alpha \approx 0.1$ is the learning rate.

To see how this equation arises, consider the following. Let us assume we are in state s, have selected action a and thus will experience k more rewards en route to the goal state. Thus:

$$Q(s, a) \approx r_1 + r_2 + r_3 + ... + r_k. \quad (3)$$

Now consider that as a result of taking action a from state s we receive our immediate reward (the r_1 above) and then transition to the next state s'. The Q-values in the table for state s' indicate our expected sum of future rewards when starting from this state. These should be all the rewards we have listed above except for the one we just received (r_1). Specifically for some action a' we have:

$$Q(s', a') \approx r_2 + r_3 + ... + r_k. \quad (4)$$

Substituting the two equations we arrive at

$$Q(s, a) = r_1 + Q(s', a'). \quad (5)$$

Remember that each Q entry is an estimate of the true value of the state/action pair. Since states that are closer to the goal are typically more accurate, $r_1 + Q(s', a')$ is probably a more accurate estimate of expected future rewards than is $Q(s, a)$. We use the difference between these two values, the *temporal difference error*, to update the value for $Q(s, a)$. Thus we arrive at the basic temporal difference update step of Equation 1.

3 Multi-Agent Sharing Algorithms

Since the performance of RL algorithms depends upon the agent's experience, we should be able to accelerate the learning process by sharing experience among agents who are all interacting with the same environment. Before we embark on a discussion of specific algorithms, we must first clearly state what is meant by "accelerating performance" so that we have a quantifiable objective.

There are two basic dependent variables that can be used to measure the speed of a learning algorithm:

1. By number of trials.

2. By wall clock time.

These two metrics are synonymous for single agent systems but will differ in a parallel implementation MAS.

The first dependent variable assesses an agent's performance as a function of the amount of learning experience. An agent's action choice and subsequent move to another state counts as one unit of experience. A sequence of actions from the start state to the goal state is one trial (also called a trajectory or episode). The trial is a convenient point in the learning process to assess the agent's performance. We can use a count of the number of trials it takes an agent to reach a specified performance metric as the *time* the agent needs to learn.

The second possible dependent variable is elapsed time, or wall clock time. Here the objective is to reach a given level of learning proficiency as quickly as possible. Naturally this metric will vary depending upon hardware platform, but for a fixed platform, we can make algorithmic choices to speed up the process.

In this paper, we focus on the first metric of learning-per-trial to assess performance. Though improved wall-clock time will ultimately be the holy grail of a parallel implementation, here we first seek an algorithm that allows our agents to collectively gain the most information from the available training data. Because we will eventually implement these algorithms on parallel hardware, we temper our algorithmic choices with realistic expectations about computing time. Though there are algorithms that employ enormous amounts of processor time and/or memory in order to achieve optimal learning speed on a per-trial basis, we do not consider these choices. Our goal is to retain the same general assumptions about realistic individual reinforcement learners and combine their experience in a computationally tractable way so as to improve the learning of the group as a whole.

Figure 1 depicts the central idea behind homogenous multi-agent reinforcement learning. The algorithms alternate steps of individual agent learning (Q-Learning) with episodes of inter-agent sharing.

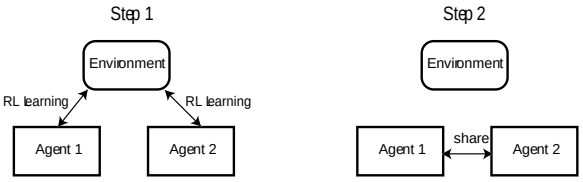

Figure 1: Sharing Experience

There are several variables that can be altered within this framework:

- **Number of agents:**
  We vary the number of agents from one (single agent reinforcement learning) to ten.

- **Frequency of Sharing:**
  The frequency of sharing refers to the number of learning trials that elapse in Step 1 before the agents share experience. We show that more frequent sharing improves learning speed.

- **Type of Information Shared:**
  We study how individual agents can share portions of their Q-values though there are many other types of sharing (environmental perceptions, recommendations, raw experience, etc.).

- **Timing of Update:**
  Updates can be synchronous where all agents share information simultaneously, or asynchronous where agents only share at irregular

intervals and they only share with "available neighbors". Here we only consider the fully synchronous case.

3.1 Why is homogenous multi-agent reinforcement learning hard?

In an ideal situation, ten agents working in parallel would collectively learn ten times as fast. We refer to this ideal situation as *linear speed-up*: n agents learn with $\frac{1}{n}$ fewer trials. In practice, linear speed-up is not a realistic possibility.

The root of the difficulty lies in the contention between several key factors. The primary reason for not achieving linear speed-up is *duplication* of learning: some of an agent's learning experience will be discarded or wasted because other agents are learning the same thing. Intuitively we would like agents to communicate as often as possible to minimize the amount of learning overlap. We would also like to have large amounts of memory and data exchange in order to not lose any information in the sharing process.

However, each of these requirements cannot be met in many realistic learning situations. The first requirement of frequent communication necessitates high bandwidth. In an actual hardware implementation, the time penalty for communication between processors (ie agents) will severely limit the communication rate. Our second request for large information stores may also not be practical for individual processor (agent) memories and especially for information transfer (again bandwidth) between agents. We will have to settle on a happy medium of occasional communication and limited information storage/exchange.

3.2 Two Algorithms

We present two different sharing algorithms. While there are many dozens of RL sharing algorithms, we present two relatively simple ones and evaluate their performance.

Constant Share RL

In Constant Share RL we assume access to unlimited bandwidth and thus have the agents communicate after every single move. We assume that there is a common Q-value table which all the agents can access and update. Each agent takes one move and then updates their corresponding Q-value. This algorithm is probably unrealistic for porting to parallel hardware as the penalty for shared memory will overwhelm any performance gains from multiple agents. However in certain situations, such as learning on a slower real-time system, this type of algorithm could be realistically implementable.

Max Share RL

Here we make the assumption that agents communicate much less frequently (perhaps every 10^4 trials or so). Each agent will have to store its own experience in Q-value form and then share the Q-values. Every agent maintains a series of counters, one counter per Q-value. Each time the agent updates its Q-value table, it also increments its own counter. At the time for sharing, each Q-value is "voted" upon by the collection of agents. The agent with the most experience (largest counter) for that Q-value wins the vote and its Q-value entry is shared among the whole population.

This algorithm has the nice feature that the agent with the most experience wins; it is reasonable to

assume that this agent has the best estimate of the Q-value. However, it is likely that many other agents also have learning experience with this Q-value. Their experience is completely discarded and hence is "wasted" computation. We would expect this algorithm to have significantly less than the ideal linear speed-up.

4 An Example Task

We turn to a two-person childhood guessing game as an initial test for our multi-agent learning algorithms. The *leader* thinks of a secret number between 1 and 20 inclusive. The *guesser* then makes guesses at the secret number. After each guess, the leader tells the guesser if their guess was (i) correct, in which case the game ends, (ii) too high, or (iii) too low. The guesser continues until they narrow in on the secret number. The objective of the guesser is to arrive at the secret number using as few guesses as possible.

4.1 Game Implementation

For machine learning purposes, the environment plays the role of the leader while the individual agents each assume the role of guesser. We extend the game to numbers 1 through 100 to increase the state space and hence increase the learning effort.

Hopefully, most computer science freshmen will quickly see that the optimal guessing strategy is a binary search tactic in which we guess the middle number between the current lower and upper bounds; without the benefit of an undergraduate education in computer science, our RL agents must learn this optimal strategy by trial and error.

Each state of the environment is composed of an ordered pair (lb, ub) where lb is the current possible lower bound and ub is the current possible upper bound. The starting state is $s = (1, 100)$ to indicate that all numbers between one and one hundred are initially possible. Suppose the agent guesses 50 and the environment responds "lower"; the next state is then $(1, 49)$ to indicate the change in upper bound. There are $\frac{100 \cdot 101}{2} + 1 = 5051$ possible states where the extra state was added as the goal state which is reached after the secret number has been guessed (necessary for the implementation of RL). The set of legal actions for each state is any number a such that $lb \leq a \leq ub$. Notice the set of action choices is different for each state and also that we do not permit our agent to guess outside of the upper and lower bounds (a wasted guess).

The reward signal is $r = -1$ for every possible state/action pair. This signal acts as a "negative counter" incrementing for each guess that does not reach the goal state. Thus in order to maximize the sum of rewards, the agent will be motivated to reach the goal state in as few guesses as possible to accumulate the fewest number of negative rewards. This type of reward signal is common for tasks in which the objective is to reach the goal in the fewest number of steps. A quick calculation shows that by following the optimal guessing strategy (binary search), an agent will arrive at the correct number after an average of 5.8 guesses; that is, the Q-value for the starting state, $Q(s, a)$, is -5.8 for $s = (1, 100)$ and $a = 50$.

4.2 Evaluation

Each of the learning algorithms is permitted to operate for a specified number of trials (games). Then we extract the current policy from the agent(s)

and evaluate this policy to ascertain the learning progress of the algorithm.

Policy extraction is relatively straightforward. For each state, s, we scan through the set of available actions to find the maximum Q-value. Keep in mind this is the *estimated* optimal action which is based upon the Q-values learned to date; if the Q-values are not correct, the actions will not be correct.

Policy evaluation is a bit more tricky. Once we have an agent's policy, we can simulate a game by following the policy until we guess the secret number. We must compute the required number of guesses for each possible secret number assuming all secret numbers, 1..100, are equally probable. We average the number of guesses over all 100 secret numbers and use this result as a metric of our policy's performance.

For each of the algorithms, we compute the number of trials that elapse before the agent discovers the optimal policy of 5.8 guesses on average. We repeat the experiment multiple times and average the number of trials over all experiments to arrive at a metric that allows us to effectively compare the learning efficiency of the different multi-agent RL algorithms.

4.3 Learning Algorithm Results

Experiment 1: Learning Improvement

In this first experiment we show the performance improvement of a single RL agent learning the 100-guessing game. As can be seen from Figure 2, the agent quickly reduces the average number of guesses from about 50 (a linear search guessing strategy?) to about 20. It then takes the agent more extensive experience to make further small improvements to its policy; it must "find" the correct policy for not only the most common states but also the more obscure ones. On average, it takes a single agent about 2.2 million trials to finally learn the perfect policy and arrive at the 5.8 average guess mark. Only the first 1000 trials are shown on this graph.

Figure 2: Performance vs Learning Experience

Details:

We run 100 experiments each consisting of an agent that is policy-evaluated after every 10 trials of experience for a total of 1000 trials. The agent uses standard Q-learning with $\alpha = 0.1$, $\epsilon = 0$ (ϵ-greedy policy), $\gamma = 0$.

Experiment 2: Speed-up vs Number Agents

Next we examine how the learning speed can be improved by adding more agents. Here we study the speed-up effect for each algorithm: *Constant Share* in which each agent updates a universal Q-table after each step of interaction, and *Max Share* in which agents vote after every 10,000 trials on which Q-values to keep. Figure 3 shows the speed-up obtained by adding extra agents to the population.

In the Constant Share algorithm we see a surprise;

this agent learns with less than the ideal linear speed-up efficiency even though it is the most frequent mode of information exchange. From the table below, we see that one agent (single Q-learning) takes about 2.2 million trials to reach the optimal policy. Ten agents require only 296,000 trials to reach the objective but this is more than the $\frac{1}{10}$ of the 2.2 million trials of a single agent. The speed-up here is only 0.76 of the ideal linear goal.

	# Agents				
Algorithm	**1**	**2**	**3**	**5**	**10**
Constant Share	224.6	117.4	80.7	52.5	29.6
speed-up/# agents	1.0	0.96	0.93	0.86	0.76
Max Share	220.7	175.6	165.5	152.8	140.0
speed-up/# agents	1.0	0.63	0.45	0.29	0.16

Trials ($\times 10^4$) to learn optimal policy.

As expected, the speed-up for the Max Share algorithm is significantly less. Ten agents here require 1.4 million trials of experience for a speed-up of only 0.16.

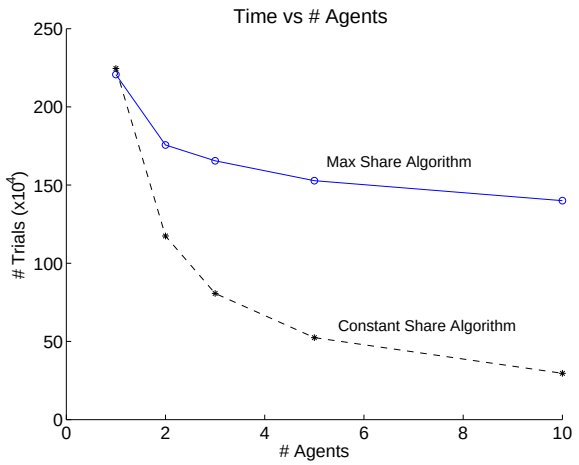

Figure 3: Learning Time vs # Agents

Details:

For each algorithm (Constant Share and Max Share), we run 50 experiments each consisting of an agent that is policy-evaluated after every 10,000 trials of experience; the results in the graph are averaged over the 50 experiments. Agents in the Max Share algorithm are permitted to share experience just prior to policy evaluation (the Constant Share agents are already using one set of Q-values). For each algorithm, we run the experiment using $n = 1, 2, 3, 5, 10$ agents. All agents use standard Q-learning with $\alpha = 0.1$, $\epsilon = 0$ (ϵ-greedy policy), $\gamma = 0$.

Experiment 3: Speed-up vs Sharing Rate

In this third experiment we confirm our belief that sharing information more frequently will expedite the learning process. For the Max Share algorithm with $n = 5$ agents, we study the performance gained by reducing the number of trials between trials of sharing. In Figure 4 we see that sharing every 100 trials requires 720 thousand trials to reach the optimal policy (compare to 1.51 million trials for sharing every 10,000).

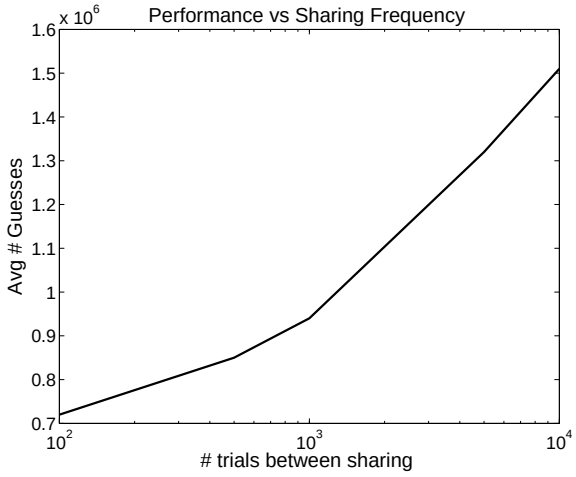

Figure 4: Learning Time vs Sharing Frequency

Details: We run the Max Share algorithm for $n = 5$ agents using

intervals of $10k$, $5k$, $1k$, 500, and 100 trials between sharing periods. Each experiment is run 30 times and the results are averaged. All agents use standard Q-learning with $\alpha = 0.1$, $\epsilon = 0$ (ϵ-greedy policy), $\gamma = 0$.

5 Concluding Remarks and Future Work

Here we have shown that homogeneous MAS can expedite the learning process via information exchange. However, our results indicate that *efficient* speed-up is difficult to obtain. Experiments on each of the algorithms, Constant Share and Max Share, have some sobering conclusions about the potential for speed-up on a parallel platform.

First consider the Constant Share algorithm in our second experiment in which we compare the speed-up achieved by varying the number of agents in the population (1, 2, 3, 5, 10). For the Constant Share algorithm, where we would most expect nearly ideal linear speed-up, we fall far short of the goal. Ten agents are only 7.6 times more efficient than a single agent (not 10 times more efficient). Why? We hypothesize that the MAS encounters learning overlap problems even at this fine-grained level of sharing.

Consider what happens when two agents embark on learning all at once (parallel case) vs what happens when an agent does two consecutive trials (serial case). In the serial case, the second trial can utilize all the Q-values that were updated during the first trial. But in the parallel case, the Q-values along the trajectory to the goal state have not yet been updated; both agents are ”flying blind” in the state space. This has two effects: first the Q-values do not get updated in the same way and second the state space is searched less efficiently (more duplication). Keep in mind that the Constant Share algorithm is most likely not realistically implementable on parallel hardware.

For the more realistic Max Share algorithm, the speed-up is even less impressive. Ten agents obtain only 1.6 times linear speed-up over a single agent (instead of 10 times). The situation improves only slightly when we increase the frequency of information exchange as in the third experiment.

These results point to considerations when we move toward an implementation on a parallel system. First, massively parallel implementations with hundreds of processors are probably not realistic. Our results show that as the number of agents is increased the speed-up drops quickly probably due to an increase in wasted learning overlap. Second, any sharing algorithm will have to be tuned to the particulars of the parallel computing hardware. We reduce the learning time by having multiple agents in parallel. But these agents must communicate and doing so incurs a time penalty that depends upon the bandwidth of the system.

These initial results point to obvious avenues for future work. Primarily, there are many other sharing algorithms that we have not considered here. Some of the others may be more efficient at exchanging experience and directing non-overlapping search strategies. Also we need to test the algorithm(s) on different tasks. It may be that the 100 guessing game is particularly suited to some types of sharing algorithms but not others. It seems reasonable that tasks with more sparse state spaces would benefit even more from a parallel implementation because of the reduced likelihood of overlapped learning. Finally there is the obvious step of porting these algorithms to actual parallel hardware. Here we will be able to study the true trade-offs of communication

overhead vs experience sharing.

References

[1] J. Andrew Bagnell. A robust architecture for multiple-agent reinforcement learning. Master's thesis, University of Florida, 1998.

[2] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. AAAI, 1998.

[3] Guillaume Laurent and Emmanuel Piat. Parallel q-learning for a block-pushing problem. In *Proceedings of the International Conference on Intelligent Robots and Systems, IROS 2001*, pages 286–291, 2001.

[4] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, 1994.

[5] Michael L. Littman. Value-function reinforcement learning in markov games. *Journal of Cognitive Systems Research*, 2001.

[6] Manisha Mundhe and Sandip Sen. Evaluating concurrent reinforcement learners. In *Proceedings of the Fourth International Conference on Multiagent Systems*, pages 421–422. IEEE Press, 2000.

[7] Luis Nunes. On learning by exchanging advice. In *Second Symposium on Adaptive Agents and Multi-Agent Systems, AAMAS-II*, 2002.

[8] Alicia Printista, Marcelo Errecalde, and Cecilia Montoya. A parallel implementation of q-learning based on communication with cache. *Journal of Computer Science and Technology*, 1(6), May 2002.

[9] Brian Sallans and Geoffrey Hinton. Using free energies to represent q-values in a multiagent reinforcement learning task. In *Advances in Neural Information Processing Systems 13 (NIPS 2001)*, volume 13. MIT Press, 2001.

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[11] Ming Tan. Multi-agent reinforcement learning: Independent and cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

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

[13] Gerhard Weiss and Pierre Dillenbourg. *Collaborative-learning: Cognitive and Computational Approaches*, chapter What is 'multi' in multi-agent learning? Pergamon Press, 1999.