Individual learning of coordination knowledge

Sandip Sen & Mahendra Sekaran
Department of Mathematical & Computer Sciences
University of Tulsa
600 South College Avenue
Tulsa, OK 74104-3189
Phone: +918-631-2985
e-mail:sandip@kolkata.mcs.utulsa.edu

Abstract

Social agents, both human and computational, inhabiting a world containing multiple active agents, need to coordinate their activities. This is because agents share resources, and without proper coordination or "rules of the road", everybody will be interfering with the plans of others. As such, we need coordination schemes that allow agents to effectively achieve local goals without adversely affecting the problem-solving capabilities of other agents. Researchers in the field of Distributed Artificial Intelligence (DAI) have developed a variety of coordination schemes under different assumptions about agent capabilities and relationships. Whereas some of these research have been motivated by human cognitive biases, others have approached it as an engineering problem of designing the most effective coordination architecture or protocol. We evaluate individual and concurrent learning by multiple, autonomous agents as a means for acquiring coordination knowledge. We show that a uniform reinforcement learning algorithm suffices as a coordination mechanism in both cooperative and adversarial situations. Using a number of multiagent learning scenarios with both tight and loose coupling between agents and with immediate as well as delayed feedback, we demonstrate that agents can consistently develop effective policies to coordinate their actions without explicit information sharing. We demonstrate the viability of using both the Q-learning algorithm and genetic algorithm based classifier systems with different payoff schemes, namely the bucket brigade algorithm (BBA) and the profit sharing plan (PSP), for developing agent coordination on two different multi-agent domains. In addition, we show that a semi-random scheme for action selection is preferable to the more traditional fitness proportionate selection scheme used in classifier systems.

1 Introduction

One of the primary goals of researchers in the artificial intelligence (AI) community is to develop autonomous agents that are knowledgeable and cognizant enough to carry out at least routine activities performed by humans. The problem, however, is an extremely difficult one, and decades of research on related areas have only served to highlight its complexity and magnitude. While researchers are developing interesting results in crucial areas like knowledge representation, planning, learning, non-monotonic reasoning, cooperative problem solving etc., it is equally important to analyze and experiment with results from one such subfield to benefit one or more of the other ones. Isolated, and domain-specific developments in any one of the sub-fields of AI is not going to go a long way to advancing the whole field.

In this paper, we will demonstrate how noteworthy advances in one particular sub-field of AI can be effectively used in another subfield to provide agents with knowledge required to solve a difficult problem. We will be applying recent research developments from the reinforcement learning literature to the coordination problem in multiagent systems. In a reinforcement learning scenario, an agent chooses actions based on its perceptions, receives scalar feedback based on past actions, and is expected to develop a mapping from perceptions to actions that will maximize feedback. Multiagent systems are a particular type of distributed AI system [2, 29], in which autonomous intelligent agents inhabit a world with no global control or globally consistent knowledge. In contrast to cooperative problem solvers [13], agents in multiagent systems are not pre-disposed to help each other out with all the resources and capabilities that they possess. These agents may still need to coordinate their activities with others to achieve their own local goals. They could benefit from receiving information about what others are doing or plan to do, and from sending them information to influence what they do.

Whereas previous research on developing agent coordination mechanisms focused on off-line design of agent organizations, behavioral rules, negotiation protocols, etc., it was recognized that agents operating in open, dynamic environments must be able to flexibly adapt to changing demands and opportunities [29, 44, 54]. In particular, individual agents are forced to engage with other agents which have varying goals, abilities, composition, and lifespan. To

effectively utilize opportunities presented and avoid pitfalls, agents need to learn about other agents and adapt local behavior based on group composition and dynamics. For machine learning researchers, multiagent learning problems are challenging, because they violate the stationary environment assumptions used by most machine learning systems. As multiple agents learn simultaneously, the feedback received by the same agent for the same action varies considerably. The stationary environment assumption used by most current machine learning systems are not well-suited for such rapidly changing environments.

In this paper, we discuss how reinforcement learning techniques for developing policies to optimize environmental feedback, through a mapping between perceptions and actions, can be used by multiple agents to learn coordination strategies without having to rely on shared information. These agents work in a common environment, but are unaware of the capabilities of other agents and may or may not be cognizant of goals to achieve. We show that through repeated problem-solving experience, such agents can develop policies to maximize environmental feedback that can be interpreted as goal achievement from the viewpoint of an external observer. More interestingly, we demonstrate that in some domains these agents develop policies that complement each other.

To evaluate the applicability of reinforcement learning schemes for enabling multiagent coordination, we chose to investigate a number of multiagent domains with varying environmental characteristics. In particular, we designed environments in which the following characteristics were varied:

Agent coupling: In some domains the actions of one agent strongly and frequently affect the plans of other agents (tightly coupled system), whereas in other domains the actions of one agent only weakly and infrequently affect the plans of other agents (loosely coupled system).

Agent relationships: Agents in a multiagent system can have different kinds of mutual relationships:

- they may act in a group to solve a common problem (cooperative agents),
- they may not have any preset dispositions towards each other but interact because they use common resources (indifferent agents),

- they may have opposing interests (adversarial agents).

For discussions in this paper, we have group the latter two class of domains as non-cooperative domains.

Feedback timing: In some domains, the agents may have immediate knowledge of the effects of their actions, whereas in others they may get the feedback for their actions only after a period of delay.

Optimal behavior combinations: How many behavior combinations of participating agents will optimally solve the task at hand? This value varies from one to infinite for different domains.

In addition, in this paper we concentrate exclusively on domains in which agents have little or no pre-existing domain expertise, and have no information about the capabilities and goals of other agents. These assumptions make the coordination problem particularly hard. This is particularly evident from the fact that almost all currently used coordination mechanisms rely heavily on domain knowledge and shared information between agents. The goal of our work is not to replace the previously developed coordination schemes, but to complement them by providing new coordination techniques for domains where the currently available schemes are not effective. In particular, domains where agents know little about each other provide a difficult challenge for currently used coordination schemes. Our contention is that problem solving performance or the feedback received from the environment can be effectively used by reinforcement based learning agents to circumvent the lack of common knowledge.

To verify our intuitions more rigorously, we decided to investigate two well-known reinforcement learning schemes: the Q-learning algorithm developed by Watkins [51], and the classifier systems method developed by Holland. Whereas the Q-learning algorithm was inspired by the theory of dynamic programming for optimization, classifier systems arose from an interesting blend of rule-based reasoning and computational mechanisms inspired by natural genetics [4, 25]. Q-learning and classifier systems have been proposed as two general reinforcement learning frameworks for achieving agent coordination in a multiagent

system. This research opens up a new dimension of constructing coordination strategies for multiagent systems.

At this point, we would like to emphasize the difference between this work and other recent publications in the nascent area of multiagent learning (MAL) research. Previous proposals for using learning techniques to coordinate multiple agents have mostly relied on using prior knowledge [5], or on cooperative domains with unrestricted information sharing [47]. A significant percentage of this research have concentrated on cooperative learning between communicating agents where agents share their knowledge or experiences [16, 38, 37, 50]. Some researchers have used communication to aid agent groups jointly decide on their course of actions [52]. The isolated instances of research in MAL that do not use explicit communication have concentrated on competitive, rather than cooperative, domains [7, 30, 40]. We strongly believe that learning coordination policies without communication has a significant role to play in competitive as well as cooperative domains. There is little argument over the fact that communication is an invaluable tool to be used by agent groups to coordinate their activities. At times communication is the most effective and even perhaps the only mechanism to guarantee coordinated behavior. Though communication is often helpful and indispensable as an aid to group activity, it does not guarantee coordinated behavior [22], is time-consuming and can detract from other problem solving activity if not carefully controlled [12]. Also, agents overly reliant on communication will be severely affected if the quality of communication is compromised (broken communication channels, incorrect or deliberately misleading information, etc.). At other times communication can be risky or even fatal (as in some combat situations where the adversary can intercept communicated messages).

We believe that even when communication is feasible and safe, it may be prudent to use it only as necessary. For example, if an agent is able to predict the behavior of other agents from past observations, it can possibly adjust its own plans to use a shared resource without the need for explicitly arriving at a contract using communication that consumes valuable time (to have performance guarantees, though, contracts arrived at using communication is possibly the most effective procedure; most of the problem solving activities by agents, however, do not involve hard guarantees or deadlines). We should strive to realize

the maximum potential of the system without using communication. Once that has been accomplished, communication can be added to augment the performance of the system to the desired efficiency level. Such a design philosophy will lead to systems where agents do not flood communication channels with unwarranted information and agents do not have to shift through a maze of useless data to locate necessary and time-critical information. With this goal in mind we have investigated the usefulness of acquiring coordination strategies without sharing information [43, 45]. We expand on this body of work in this paper, and explore the advantages and limitations of learning without communication as a means to generating coordinated behavior in autonomous agents.

The rest of the paper is organized as follows: Section 2 presents highlights of the previous approaches to developing coordination strategies for multiple, autonomous agents; Section 3 provides a categorization of multiagent systems to identify different learning scenarios and presents a sampling of prior multiagent learning research; Section 4 reviews the reinforcement learning techniques that we have utilized in this paper; Sections 5, 6, and 7 present results of experiments with Q-learning and genetic algorithm based reinforcement learning systems on a block pushing, robot navigation, and resource sharing domains respectively; Section 8 summarizes the lessons learnt from this research and outlines future research directions.

2 Coordination of multiple agents

In a world inhabited by multiple agents, coordination is a key to group as well as individual success. We need to coordinate our actions whenever we are sharing goals, resources, or expertise. By coordination we mean choosing one's own action based on the expectation of others' actions. Coordination is essential for cooperative, indifferent, and even adversarial agents. As computer scientists, we are interested in developing computational mechanisms that are domain independent and robust in the presence of noisy, incomplete, and out-of-date information. Research in the area of multiagent systems has produced techniques for allowing multiple agents, which share common resources, to coordinate their actions so that individually rational actions do not adversely affect overall system efficiency [2, 13, 17, 26].

Coordination of problem solvers, irrespective of whether they are selfish or cooperative,

is a key issue to the design of an effective multiagent system. The search for domain-independent coordination mechanisms has yielded some very different, yet effective, classes of coordination schemes. The most influential classes of coordination mechanisms developed to date are the following:

- protocols based on contracting [9, 48]
- distributed search formalisms [14, 57]
- organizational and social laws [15, 32, 33]
- multi-agent planning [11, 39]
- decision and game theoretic negotiations [18, 19, 58]
- linguistic approaches [8, 55]

Whereas some of these work uses architectures and protocols designed off-line [15, 46, 48] as coordination structures, others acquire coordination knowledge on-line [11, 19]. Almost all of the coordination schemes developed to date assume explicit or implicit sharing of information. In the explicit form of information sharing, agents communicate partial results [11], speech acts [8], resource availabilities [48], etc. to other agents to facilitate the process of coordination. In the implicit form of information sharing, agents use knowledge about the capabilities of other agents [15, 18, 58] to aid local decision-making. Though each of these approaches has its own benefits and weaknesses, we believe that the less an agent depends on shared information, and the more flexible it is to the on-line arrival of problem-solving and coordination knowledge, the better it can adapt to changing environments. As flexibility and adaptability are key aspects of intelligent and autonomous behavior, we are interested in investigating mechanisms by which agents can acquire and use coordination knowledge through interactions with its environment (that includes other agents) without having to rely on shared information.

One can also argue that pre-fabricated coordination strategies can quickly become inadequate if the system designer's model of the world is incomplete/incorrect or if the environment in which the agents are situated can change dynamically. Coordination strategies that

incorporate learning and adaptation components will be more robust and effective in these more realistic scenarios. Thus agents will be able to take advantage of new opportunities and deal with new contingencies presented by the environment which cannot be foreseen at design time.

3 Learning in multiagent systems

To highlight learning opportunities inherent in most multiagent systems, we develop here a categorization of multiagent problems that can be used to characterize the nature of learning mechanisms that should be used for these problems. The categorization presented in Table 1 is not meant to be the only or even the most definitive taxonomy of the field. The dimensions we consider for our taxonomy are the following: *agent relationships* (cooperative vs. non-cooperative), *use of communication* (agents explicitly communicating or not). Within cooperative domains again, we further categorize domains based on *decision-making* authorities (individual vs. shared). These dimensions are not necessarily completely orthogonal, e.g., shared decision making may not be feasible without the use of explicit communication.

We use the term cooperative relationship to refer to situations where multiple agents are working towards a common goal. Though there may be local goals, these are in fact, subgoals of the common goal (even if different subgoals interfere). A number of different domains fall under the non-cooperative spectrum. These include competitive scenarios (one agent's gain is another agent's loss; need not be zero-sum), as well as scenarios where agents coordinate only to avoid conflicts.

As mentioned before, shared learning by a group of cooperative agents have received the most attention in MAL literature [16, 37, 52]. Others have looked at each agent learning individually but using communication to share pertinent information [6, 38, 50]. Learning to cooperate without explicit information sharing can be based on primarily environmental feedback [45] or from observation of other agents in action [23]. Researchers have investigated two approaches to learning without explicit communication in non-cooperative domains: (a) treating opponents as part of the environment without explicit modeling [43], (b) learning explicit competitor models [30, 40]. Little work has been done to date in learning to compete

with explicit communication. Possible scenarios to investigate in this area include learning the strengths and weaknesses of the competitor from intercepted communication, or proactively probing the opponent to gather more information about its preferences.

Even previous work on using reinforcement learning for coordinating multiple agents [50, 52] have relied on explicit information sharing. We, however, concentrate on systems where agents share no problem-solving knowledge. We show that although each agent is independently using reinforcement learning techniques to optimizing its own environmental reward, global coordination between multiple agents can emerge without explicit or implicit information sharing. These agents can therefore act independently and autonomously, without being affected by communication delays (due to other agents being busy) or failure of a key agent (who controls information exchange or who has more information), and do not have to be worry about the reliability of the information received (Do I believe the information received? Is the communicating agent an accomplice or an adversary?). The resultant systems are, therefore, robust and fault-tolerant.

Schaerf *et al.* have studied the use of reinforcement learning based agents for load balancing in distributed systems [41]. In this work, a comprehensive history of past performance is used to make informed decisions about choice of resources to submit jobs to. Parker [36] has studied the emergence of coordination in simulated robot groups by using simple adaptive schemes that alter robot motivations. A major difference from our work is that the simulated robots in this work build explicit models of other robots. Other researchers have used reinforcement learning for developing effective groups of physical robots [34, 56]. Mataric [34] concentrates on using intermediate feedback for subgoal fulfillment to accelerate learning. In contrast with our work, the evaluation of learning effectiveness under varying degrees of interaction between the agents is not the focus of this work. The work by Yanco and Stein [56] involves using reinforcement learning techniques to evolve effective communication protocols between cooperating robots. This is complementary to our approach of learning to coordinate in the absence of communication.

Though our agents can be viewed as a learning automaton, the scalar feedback received from the environment prevents the use of results directly from the field of *learning automata* [35]. Recent work on theoretical issues of multiagent reinforcement learning promises

to produce new frameworks for investigating problems such as those addressed in this paper [21, 42, 53].

4 Reinforcement learning

In reinforcement learning problems [1, 27] reactive and adaptive agents are given a description of the current state and have to choose the next action from a set of possible actions so as to maximize a scalar *reinforcement* or *feedback* received after each action. The learner's environment can be modeled by a discrete time, finite state, Markov decision process that can be represented by a 4-tuple $\langle S, A, P, r\rangle$ where $P : S \times S \times A \mapsto [0, 1]$ gives the probability of moving from state s_1 to s_2 on performing action a, and $r : S \times A \mapsto \Re$ is a scalar reward function. Each agent maintains a policy, π, that maps the current state into the desirable action(s) to be performed in that state. The expected value of a discounted sum of future rewards of a policy π at a state x is given by $V_\gamma^\pi \stackrel{\text{def}}{=} E\{\sum_{t=0}^{\infty} \gamma^t r_{s,t}^\pi\}$, where $r_{s,t}^\pi$ is the random variable corresponding to the reward received by the learning agent t time steps after if starts using the policy π in state s, and γ is a discount rate ($0 \leq \gamma < 1$).

Various reinforcement learning strategies have been proposed using which agents can develop a policy to maximize rewards accumulated over time. For evaluating the classifier system paradigm for multiagent reinforcement learning, we compare it with the Q-learning [51] algorithm, which is designed to find a policy π^* that maximizes $V_\gamma^\pi(s)$ for all states $s \in S$. The decision policy is represented by a function, $Q : S \times A \mapsto \Re$, which estimates long-term discounted rewards for each state–action pair. The Q values are defined as $Q_\gamma^\pi(s, a) = V_\gamma^{a;\pi}(s)$, where $a;\pi$ denotes the event sequence of choosing action a at the current state, followed by choosing actions based on policy π. The action, a, to perform in a state s is chosen such that it is expected to maximize the reward,

$$V_\gamma^{\pi^*}(s) = \max_{a \in A} Q_\gamma^{\pi^*}(s, a) \text{ for all } s \in S.$$

If an action a in state s produces a *reinforcement* of R and a transition to state s', then the corresponding Q value is modified as follows:

$$Q(s, a) \leftarrow (1 - \beta)\, Q(s, a) + \beta\, (R + \gamma \max_{a' \in A} Q(s', a')). \tag{1}$$

The above update rule is similar to Holland's bucket-brigade [25] algorithm in classifier systems and Sutton's temporal-difference [49] learning scheme. The similarities of Q-learning and classifier systems have been analyzed in [10].

Classifier systems are rule based systems that learn by adjusting rule strengths from feedback and by discovering better rules using genetic algorithms. In this paper, we will use simplified classifier systems where all possible message action pairs are explicitly stored and classifiers have one condition and one action. These assumptions are similar to those made by Dorigo and Bersini [10]; we also use their notation to describe a classifier i by (c_i, a_i), where c_i and a_i are respectively the condition and action parts of the classifier. $S_t(c_i, a_i)$ gives the strength of classifier i at time step t. We first describe how the classifier system performs and then discuss two different feedback distribution schemes, namely the Bucket Brigade algorithm (BBA), and the Profit Sharing Plan (PSP).

All classifiers are initialized to some default strength. At each time step of problem solving, an input message is received from the environment and matched with the classifier rules to form a matchset, $\mathcal{M}$. One of these classifiers is chosen to fire and based on its action, a feedback may be received from the environment. Then the strengths of the classifier rules are adjusted. This cycle is repeated for a given number of time steps. A series of cycles constitute a *trial* of the classifier system. In the BBA scheme, when a classifier is chosen to fire, its strength is increased by the environmental feedback. But before that, a fraction α of its strength is removed and added to the strength of the classifier who fired in the last time cycle. So, if classifier i fires at time step t, produces external feedback of R, and classifier j fires at the next time step, the following equations gives the strength update of classifier i:

$$S_{t+1}(c_i, a_i) = (1 - \alpha) * S_t(c_i, a_i) + \alpha * (R + S_{t+1}(c_j, a_j)).$$

We now describe the profit sharing plan (PSP) strength-updating scheme [20] used in classifier systems. In this method, problem solving is divided into episodes in between receipts of external reward. A rule is said to be active in a period if it fired in at least one of the cycles in that episode. At the end of episode e, the strength of each active rule i in that episode is updated as follows:

$$S_{e+1}(c_i, a_i) = S_e(c_i, a_i) + \alpha * (R_e - S_e(c_i, a_i)),$$

where R_e is the external reward received at the end of the episode. We have experimented with two methods of choosing a classifier to fire given the matchset. In the more traditional method, a classifier $i \in \mathcal{M}$ at time t is chosen with a probability given by $\frac{S_t(c_i,a_i)}{\sum_{d\in\mathcal{M}} S_t(c_d,a_d)}$. We call this fitness proportionate PSP or PSP(FP). In the other method of action choice, the classifier with the highest fitness in $\mathcal{M}$ is chosen 90% of the time, and a random classifier from $\mathcal{M}$ is chosen in the rest 10% cases (Mahadevan uses such an action choosing mechanism for Q-learning in [31]). We call this a semi-random PSP or PSP(SR).

For the Q-learning algorithm, we stop a run when the algebraic difference of the policies at the end of neighboring trials is below a threshold for 10 consecutive trials. With this convergence criterion, however, the classifier systems ran too long for us to collect reasonable data. Instead, every 10th trial, we ran the classifier system (both with BBA and PSP) with a deterministic action choice over the entire trial. We stopped a run of the classifier system if the differences of the total environmental feedback received by the system on neighboring deterministic trials were below a small threshold for 10 consecutive deterministic trials.

We perform an in-depth study of the effectiveness of using Q-learning by concurrent learners to develop effective coordination in a block pushing domain. We also compare the performance of classifier systems and Q-learning on a resource sharing and a robot navigation domain. The characteristics of these three domains are as follows:

Block pushing: Concurrent learning by two agents with immediate environmental feedback; strongly coupled system; multiple optimal behaviors.

Resource sharing: One agent learning to adapt to another agent's behavior with delayed environmental feedback; strongly coupled system; single optimal behavior.

Robot navigation: Concurrent learning by multiple agents with immediate environmental feedback; variable coupling; multiple optimal behaviors.

5 Block pushing problem

In this problem, two agents, a_1 and a_2, are independently assigned to move a block, b, from a starting position, S, to some goal position, G, following a path, P, in Euclidean space.

The agents are not aware of the capabilities of each other and yet must choose their actions individually such that the joint task is completed. The agents have no knowledge of the system physics, but can perceive their current distance from the desired path to take to the goal state. Their actions are restricted as follows: agent i exerts a force $\vec{F_i}$, where $0 \leq |\vec{F_i}| \leq F_{max}$, on the object at an angle θ_i, where $0 \leq \theta \leq \pi$. An agent pushing with force $\vec{F}$ at angle θ will offset the block in the x direction by $|\vec{F}|\cos(\theta)$ units and in the y direction by $|\vec{F}|\sin(\theta)$ units. The net resultant force on the block is found by vector addition of individual forces: $\vec{F} = \vec{F_1} + \vec{F_2}$. We calculate the new position of the block by assuming unit displacement per unit force along the direction of the resultant force. The new block location is used to provide *feedback* to the agent. If (x, y) is the new block location, $P_x(y)$ is the x-coordinate of the path P for the same y coordinate, $\Delta x = |x - P_x(y)|$ is the distance along the x dimension between the block and the desired path, then $K * a^{-\Delta x}$ is the feedback given to each agent for their last action (we have used $K = 50$ and $a = 1.15$).

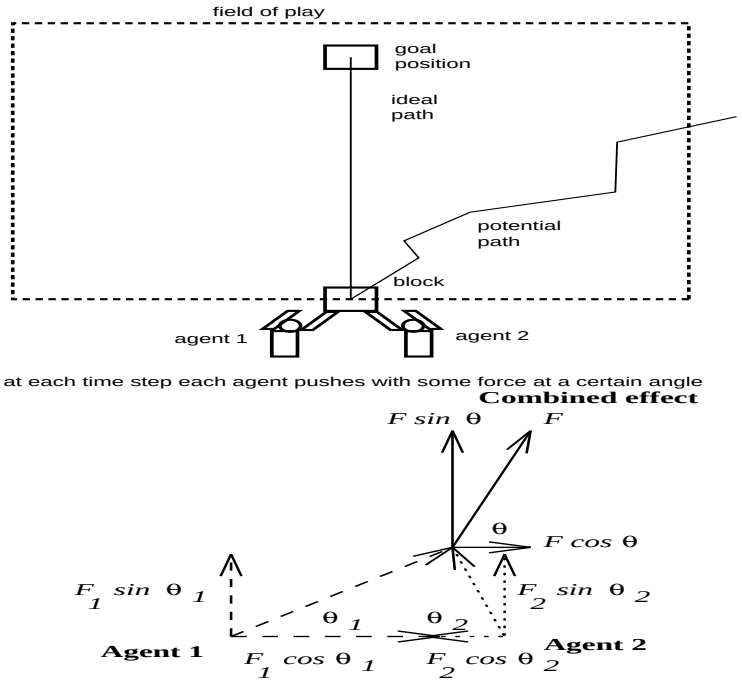

Figure 1: The block pushing problem

The field of play is restricted to a rectangle with endpoints $[0, 0]$ and $[100, 100]$. A trial consists of the agents starting from the initial position S and applying forces until either the goal position G is reached or the block leaves the field of play (see Figure 1). We abort a

trial if a pre-set number of agent actions fail to take the block to the goal. This prevents agents from learning policies where they apply no force when the block is resting on the optimal path to the goal but not on the goal itself. The agents are required to learn, through repeated trials, to push the block along the path P to the goal. Although we have used only two agents in our experiments, the solution methodology can be applied without modification to problems with arbitrary number of agents.

To implement the policy π we chose to use an internal discrete representation for the external continuous space. The force, angle, and the space dimensions were all uniformly discretized. When a particular discrete force or action is selected by the agent, the middle value of the associated continuous range is used as the actual force or angle that is applied on the block.

The Block-Pushing problem is a *tightly-coupled* system, where, at each time step the outcome of each agent action is dependent on the action of the other agents present in the system. Using the block-pushing problem, we have conducted experiments on both *cooperative* (agents push the block towards a common goal) and *non-cooperative* (agents vie with each other to push the block to individual goals) situations.

An experimental run consists of a number of trials during which the system parameters (β, γ, and K) as well as the learning problem (granularity, agent choices) is held constant. The stopping criteria for a run is either that the agents succeed in pushing the block to the goal in N consecutive trials (we have used $N = 10$) or that a maximum number of trials (we have used 1500) have been executed. The latter cases are reported as non-converged runs.

The standard procedure in Q-learning literature of initializing Q values to zero is suitable for most tasks where non-zero feedback is infrequent and hence there is enough opportunity to explore all the actions. Because a non-zero feedback is received after every action in our problem, we found that agents would follow, for an entire run, the path they take in the first trial. This is because they start each trial at the same state, and the only non-zero Q-value for that state is for the action that was chosen at the start trial. Similar reasoning holds for all the other actions chosen in the trial. A possible fix is to choose a fraction of the actions by random choice, or to use a probability distribution over the Q-values to choose actions stochastically. These options, however, lead to very slow convergence. Instead, we chose

to initialize the Q-values to a large positive number. This enforced an exploration of the available action options while allowing for convergence after a reasonable number of trials.

The primary metric for performance evaluation is the average number of trials taken by the system to converge. Information about acquisition of coordination knowledge is obtained by plotting, for different trials, the average distance of the actual path followed from the desired path. Data for all plots and tables in this paper have been averaged over 100 runs.

5.1 Experiments in Cooperative Domain

Extensive experimentation was performed on the block-pushing domain. The two agents were assigned the task of pushing the block to the same goal location. The following sections provide further details on the experimental results.

5.1.1 Choice of system parameters

If the agents learn to push the block along the desired path, the reward that they will receive for the best action choices at each step is equal to the maximum possible value of K. The steady-state values for the Q-values (Q_{ss}) corresponding to optimal action choices can be calculated from the equation:

$$Q_{ss} = (1 - \beta)\, Q_{ss} + \beta\, (K + \gamma\, Q_{ss}).$$

Solving for Q_{ss} in this equation yields a value of $\frac{K}{1-\gamma}$. In order for the agents to explore all actions after the Q-values are initialized at S_I, we require that any new Q value be less than S_I. From similar considerations as above we can show that this will be the case if $S_I \geq \frac{K}{1-\gamma}$. In our experiments we fix the maximum reward K at 50, S_I at 100, and γ at 0.9. Unless otherwise mentioned, we have used $\beta = 0.2$, and allowed each agent to vary both the magnitude and angle of the force they apply on the block.

The first problem we used had starting and goal positions at $(40, 0)$ and $(40, 100)$ respectively. During our initial experiments we found that with an even number of discrete intervals chosen for the angle dimension, an agent cannot push along any line parallel to the

y-axis. Hence we used an odd number, 11, of discrete intervals for the angle dimension. The number of discrete intervals for the force dimension is chosen to be 10.

On varying the number of discretization intervals for the state space between 10, 15, and 20, we found the corresponding average number of trials to convergence is 784, 793, and 115 respectively with 82%, 83%, and 100% of the respective runs converging within the specified limit of 1200 trials. This suggests that when the state representation gets too coarse, the agents find it very difficult to learn the optimal policy. This is because the less the number of intervals (the coarser the granularity), the more the variations in reward an agent gets after taking the same action at the same state (each discrete state maps into a larger range of continuous space and hence the agents start from and ends up in physically different locations, the latter resulting in different rewards).

5.1.2 Varying learning rate

We experimented by varying the learning rate, β. The resultant average distance of the actual path from the desired path over the course of a run is plotted in Figure 2 for β values 0.4, 0.6, and 0.8.

In case of the straight path between (40,0) and (40,100), the optimal sequence of actions always puts the block on the same x-position. Since the x-dimension is the only dimension used to represent state, the agents update the same Q-value in their policy matrix in successive steps. We now calculate the number of updates required for the Q-value corresponding to this optimal action before it reaches the steady state value. Note that for the system to converge, it is necessary that only the Q-value for the optimal action at $x = 40$ needs to arrive at its steady state value. This is because the block is initially placed at $x = 40$, and so long as the agents choose their optimal action, it never reaches any other x position. So, the number of updates to reach steady state for the Q-value associated with the optimal action at $x = 40$ should be proportional to the number of trials to convergence for a given run.

In the following, let S_t be the Q-value after t updates and S_I be the initial Q-value. Using Equation 1 and the fact that for the optimal action at the starting position, the *reinforcement* received is K and the next state is the same as the current state, we can write,

$$S_{t+1} \;=\; (1-\beta)\, S_t + \beta\, (K + \gamma\, S_t)$$

$$
\begin{aligned}
&= (1 - \beta\,(1 - \gamma))\,S_t + \beta\,K \\
&= A\,S_t + C \qquad (2)
\end{aligned}
$$

where A and B are constants defined to be equal to $1 - \beta * (1 - \gamma)$ and $\beta * K$ respectively. Equation 2 is a difference equation which can be solved using $S_0 = S_I$ to obtain

$$S_t = A^{t+1}\,S_I + \frac{C\,(1 - A^{t+1})}{1 - A}.$$

If we define convergence by the criteria that $|S_{t+1} - S_t| < \epsilon$, where ϵ is an arbitrarily small positive number, then the number of updates t required for convergence can be calculated to be the following:

$$
\begin{aligned}
t &\geq \frac{\log(\epsilon) - \log(S_I\,(1 - A) - C)}{log(A)} \\
&= \frac{\log(\epsilon) - \log(\beta) - \log(S_I\,(1 - \gamma) - K)}{log(1 - \beta\,(1 - \gamma))} \qquad (3)
\end{aligned}
$$

If we keep γ and S_I constant the above expression can be shown to be a decreasing function of β. This is corroborated by our experiments with varying β while holding $\gamma = 0.1$ (see Figure 2). As β increases, the agents take less number of trials to convergence to the optimal set of actions required to follow the desired path. The other plot in Figure 2 presents a comparison of the theoretical and experimental convergence trends. The first curve in the plot represents the function corresponding to the number of updates required to reach steady state value (with $\epsilon = 0$). The second curve represents the average number of trials required for a run to converge, scaled down by a constant factor of 0.06. The actual ratios between the number of trials to convergence and the values of the expression on the right hand side of the inequality 3 for β equal to 0.4, 0.6, and 0.8 are 24.1, 25.6, and 27.5 respectively (the average number of trials are 95.6, 71.7, and 53; values of the above-mentioned expression are 3.97, 2.8, and 1.93). Given the fact that results are averaged over 100 runs, we can claim that our theoretical analysis provides a good estimate of the relative time required for convergence as the learning rate is changed.

5.1.3 Varying agent capabilities

The next set of experiments was designed to demonstrate the effects of agent capabilities on the time required to converge on the optimal set of actions. In the first of the current set of

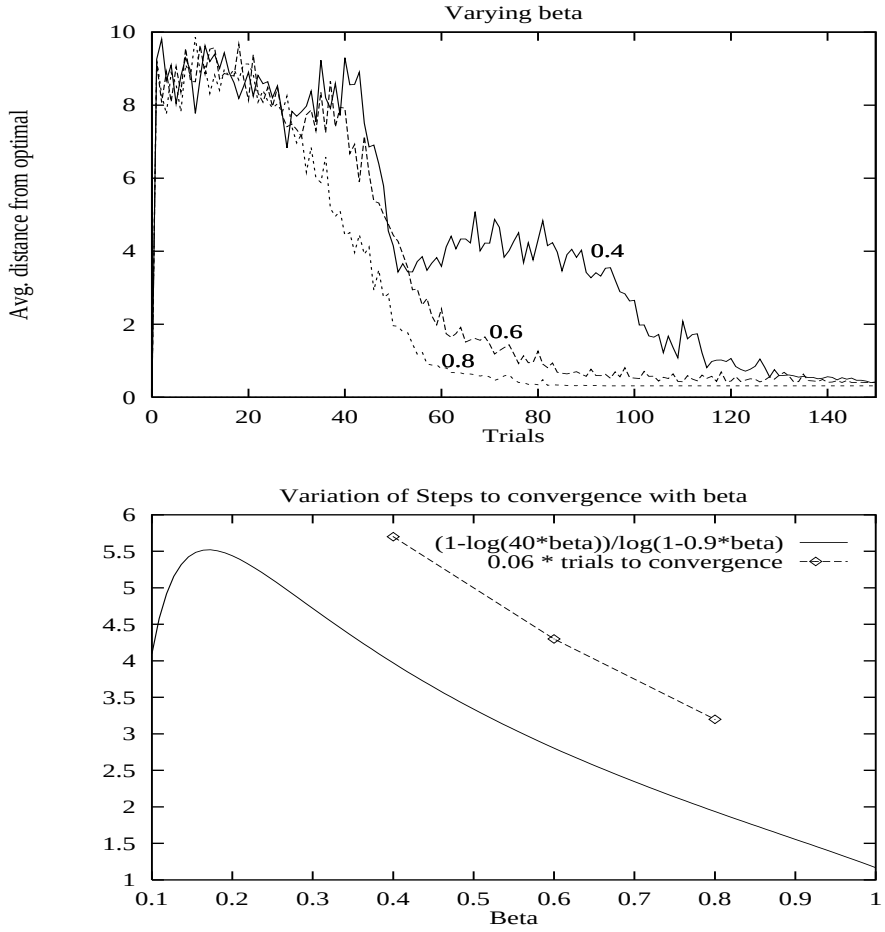

Figure 2: Variation of average distance of actual path from desired path over the course of a run, and the number of updates for convergence of optimal Q-value with changing β ($\gamma = 0.1$, $S_I = 100$).

experiments, one of the agents was chosen to be a "dummy"; it did not exert any force at all. The other agent could only change the angle at which it could apply a constant force on the block. In the second experiment, the latter agent was allowed to vary both force and angle. In the third experiment, both agents were allowed to vary their force and angle. The average number of trials to convergence for the first, second, and third experiment are 55, 431, and 115 respectively. The most interesting result from these experiments is that two agents can learn to coordinate their actions and achieve the desired problem-solving behavior much faster than when a single agent is acting alone. If, however, we simplify the problem of the only active agent by restricting its choice to that of selecting the angle of force, it can

learn to solve the problem quickly. If we fix the angle for the only active agent, and allow it to vary only the magnitude of the force, the problem becomes either trivial (if the chosen angle is identical to the angle of the desired path from the starting point) or unsolvable.

5.1.4 Transfer of learning

We designed a set of experiments to demonstrate how learning in one situation can help learning to perform well in a similar situation. The problem with starting and goal locations at (40,0) and (40,100) respectively is used as a reference problem. In addition, we used five other problems with the same starting location and with goal locations at (50,100), (60,100), (70,100), (80,100), and (90,100) respectively. The corresponding desired paths were obtained by joining the starting and goal locations by straight lines. To demonstrate transfer of learning, we first stored each of the policy matrices that the two agents converged on for the original problem. Next, we ran a set of experiments using each of the new problems, with the agents starting off with their previously stored policy matrices.

We found that there is a linear increase in the number of trials to convergence as the goal in the new problem is placed farther apart from the goal in the initial problem. To determine if this increase was due purely to the distance between the two desired paths, or due to the difficulty in learning to follow certain paths, we ran experiments on the latter problems with agents starting with uniform policies. These experiments reveal that the more the angle between the desired path and the y-axis, the longer the agents take to converge. Learning in the original problem, however, does help in solving these new problems, as evidenced by a $\approx 10\%$ savings in the number of trials to convergence when agents started with the previously learned policy. Using a one-tailed t-test we found that all the differences were significant at the 99% confidence level. This result demonstrates the transfer of learned knowledge between similar problem-solving situations.

5.1.5 Complimentary learning

In the last few sections we have shown the effects of system parameters and agent capabilities on the rate at which the agents converge on an optimal set of actions. In this section, we discuss what an "optimal set of actions" means to different agents.

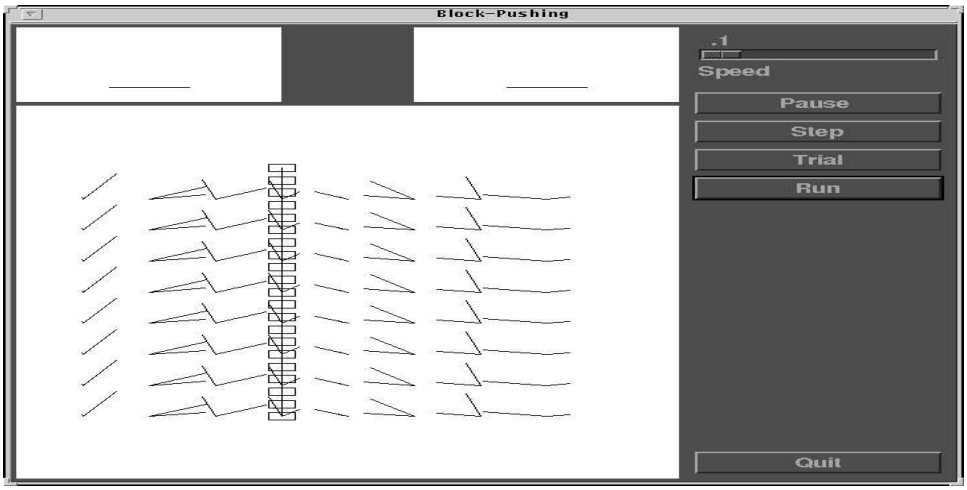

Figure 3: Optimal action choices for a selection of states for each agent according to their policy matrices at the end of a successful run.

If the agents were cognizant of the actual constraints and goals of the problem, and knew elementary physics, they could independently calculate the desired action for each of the states that they may enter. The resulting policies would be identical. Our agents, however, have no planning capacity and their knowledge is encoded in the policy matrix. Figure 3 provides a snapshot, at the end of a successfully converged run, of what each agent believes to be its best action choice for each of the possible states in the world. The action choice for each agent at a state is represented by a straight line at the appropriate angle and scaled to represent the magnitude of force. We immediately notice that the individual policies are complimentary rather than being identical. Given a state, the combination of the best actions will bring the block closer to the desired path. In some cases, one of the agents even pushes in the wrong direction while the other agent has to compensate with a larger force to bring the block closer to the desired path. These cases occur in states which are at the edge of the field of play, and have been visited only infrequently. Complementarity of the individual policies, however, are visible for all the states.

5.2 Experiments in Non-Cooperative Domains

We designed a set of experiments in which two agents are provided different feedback for the same block location. The agents are assigned to push the same block to two different

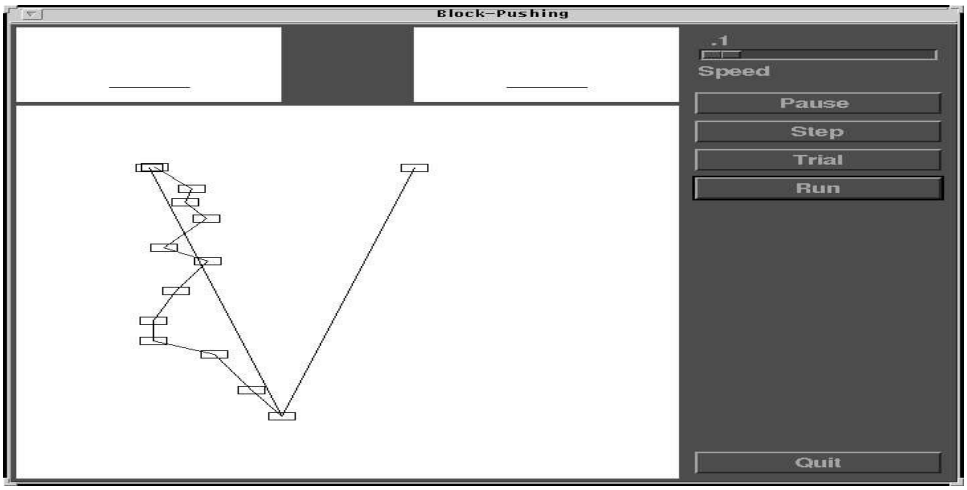

Figure 4: Example trial when agents have conflicting goals.

goals along different paths. Hence, the action of each of them adversely affects the goal achievement of the other agent. The maximum force (we refer to this as strength) of one agent was chosen as 10 units, while the maximum force of the other agent was varied. The other variable was the number of discrete action options available within the given force range. When there is considerable disparity between the strengths of the two agents, the stronger agent overpowers the weaker agent, and succeeds in pushing the block to its goal location (see Figure 4). The average number of trials to convergence (see Table 2), however, indicates that as the strength of the weaker agent is increased, the stronger agent finds it increasingly difficult to attain its goal. For these experiments, the strong and the weak agents had respectively 11 (between 0-10) and 2 (0 and its maximum strength) force options to choose from.

When the number of force discretizations for the weak agent is increased from 2 to 10, we find that the stronger agent finds it more difficult to push the block to its own goal. If we increase the maximum force of the weak agent closer to the maximum force of the stronger agent, we find that neither of them is able to push the block to its desired goal. At the of a run, we find that the final converged path lies in between their individual desired paths. As the strength of the weaker agent increases, this path moves away from the desired path of the stronger agent, and ultimately lies midway between their individual desired paths when both agents are equally strong.

Intuitively, an agent should be able to 'overpower' another agent whenever it is stronger. Why is this not happening? The answer lies in the stochastic variability of feedback received for the same action at the same state, and the deterministic choice of the action corresponding to the maximal policy matrix entry. When an agent chooses an action at a state it can receive one of several different feedbacks depending on the action chosen by the other agent. We define the optimal action choice for a state x to be the action A_x that has the highest average feedback F_x. Suppose the first time the agent chooses this action at state x it receives a feedback $F_1 < F_x$. Also, let it receive a feedback $F_2 > F_1$ for a non-optimal action A_y it chooses in the same state x. If these were the only two options available in state x, the agent would choose A_y over A_x next time it is in state x, because the former action corresponds to a higher policy matrix entry. If the steady state value of the policy matrix entry for action A_y in state x is greater than the policy matrix entry for action A_x obtained after receiving feedback F_1, the latter action will be never tried again, and hence the agent will converge on a non-optimal policy. This is a quintessential example of the *exploration-exploitation* tradeoff [24]. Also, this is more likely to happen when the same action can generate more number of distinct feedbacks (the same action for the stronger agent can produce more distinct feedbacks when the discretizations for weaker agent is increased). A simple remedy to this situation will be to choose a proportion of the actions randomly or to choose actions using a probability distribution over the policy matrix values. Each of these options, however, results in an exponential increase of the trials to convergence. Currently we are developing a simulated annealing [28] based procedure which results in a decrease in the proportion of random choices as the policy matrix converges to its steady state.

6 Robot navigation problem

We designed a problem in which four agents, A, B, C, and D, are to find the optimal path in a grid world, from given starting locations to their respective goals, A', B',C', and D'. The agents traverse their world using one of the five available operators: *north, south, east, west,* or *hold*. Figure 5 depicts potential paths that each of the agents might choose during their learning process. The goal of the agents is to learn moves that quickly take them to

their respective goal locations without colliding with other agents.

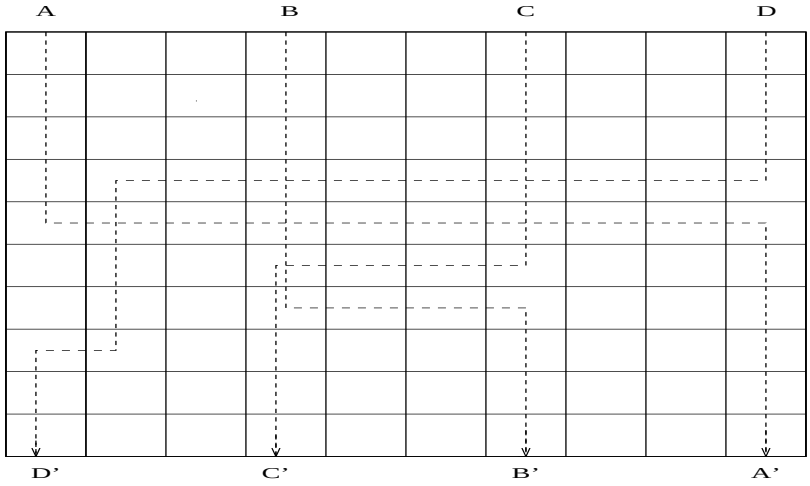

Figure 5: A robot navigation problem.

Each agent receives feedback based on its move: when it makes a move that takes them *towards* their goal, they receive a feedback of 1; when it makes a move that takes them *away* from their goal, they receive a feedback of -1; when it makes a move that results in *no change* of their distance from their goal (*hold*), they receive a feedback of 0; when it makes a move that results in a *collision*, the feedback is computed as depicted in Fig 6. All agents learn at the same time by updating their individual policies.

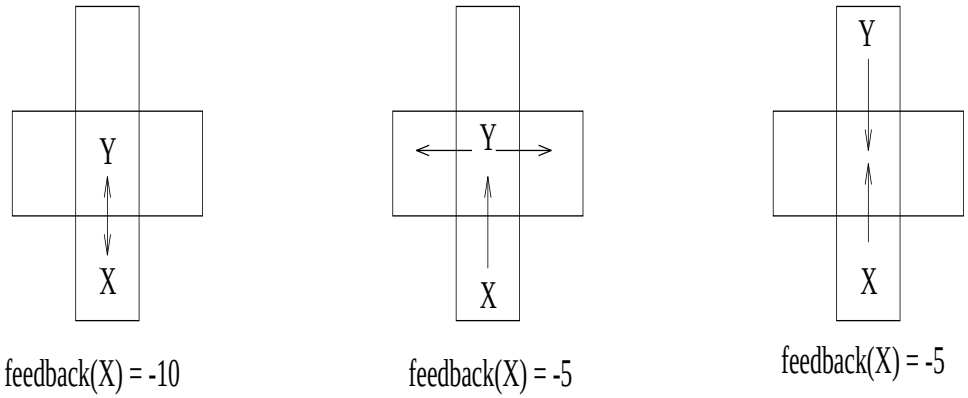

Figure 6: Feedback for agent X when it causes different types of collisions (given the action of the other agent in the collision).

The Robot navigation task is a domain in which the agent couplings varies over time. When agents are located next to each other, they are very likely to interact and hence agent

behaviors are tightly coupled. But, as they move apart, their interactions become less likely, and as a result, their behaviors become loosely coupled.

Since the robot navigation problem produces rewards after each time step, we have used the BBA method of payoff distribution with the classifier system. The system parameters are $\beta = 0.5$, and $\gamma = 0.8$ for Q-learning and $\alpha = 0.1$ for BBA.

Experimental results on the robot navigation domain comparing Q-learning and a classifier system using BBA for payoff distribution is displayed in Figure 10. Plots show the average number of steps taken by the agents to reach their goals. Lower values of this parameter means agents are learning to find more direct paths to their goals without colliding with each other. Results are averaged over 50 runs for both systems. Q-learning takes as much as 5 times longer to converge when compared to BBA. The final number of steps taken by agents using Q-learning is slightly smaller than the number of steps taken by agents using BBA. We believe that if we make the convergence criteria more strict for the BBA, a better solution can be evolved with more computational effort.

The interesting aspect of this experiment is that all the agents were learning simultaneously and hence it was not obvious that they would find good paths. Typical solutions, however, show that agents stop at the right positions to let others pass through. This avoids collisions. The paths do contain small detours, and hence are not optimal.

The performance of BBA when α=0.5 (comparable to β used in Q-learning) was not very different when compared to that presented in Figure 10. It is interesting to note that when experiments were tried setting the β in Q-learning to 0.1 (comparable to α in BBA) the system did not attain convergence even after 5000 trials.

7 Resource sharing problem

The resource sharing problem assumes two agents sharing a common resource or channel, with each of them trying to distribute their load on the system so as to achieve maximum utility. In our version of the problem, it is assumed that one agent has already applied some load distributed over a fixed time period, and the other agent is learning to distribute its load on the system without any knowledge of the current distribution. A maximum load of

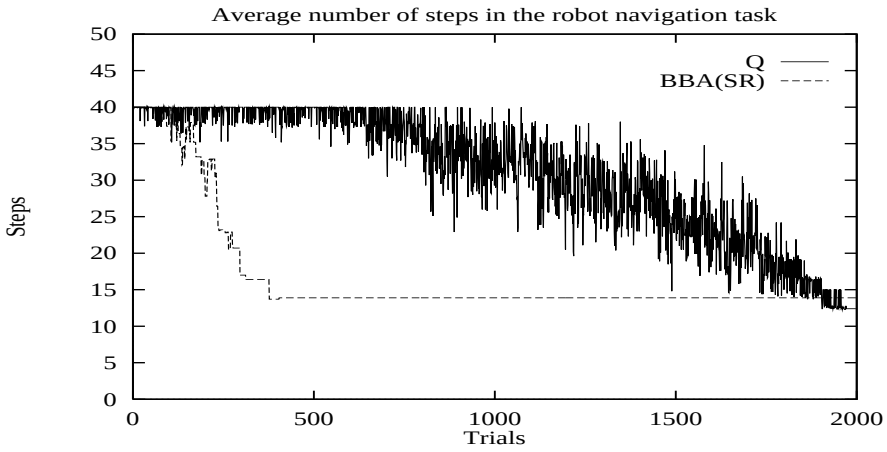

Figure 7: Comparison of BBA and Q-learning on the robot navigation problem.

L can be applied on the system at any point in time (loads in excess of this do not receive any utility).

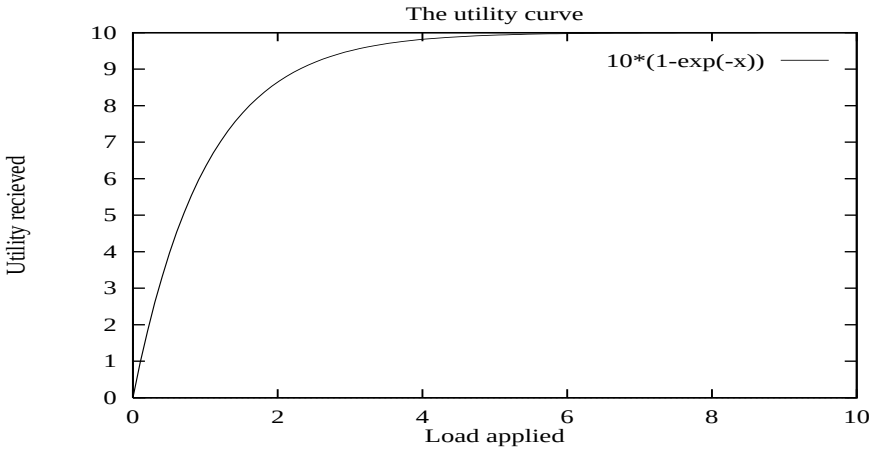

Figure 8: Curve depicting the utility received for a given load

The second agent can use K load-hours of the channel. If it applies a load of k_t load-hours on the system at time step t, when the first agent has applied l_t load-hours, the utility it receives is $u(l_t, k_t) = U(\max(L, k_t + l_t)) - U(\max(L, l_t)$, where U is the utility function in Figure 8, and $L = 10$ is the maximum load allowed on the system. The total feedback it gets at the end of T time steps is $\sum_{t=1}^{T} u(l_t, k_t)$. This problem requires the second agent to distribute its load around the loads imposed by the first agent in order to obtain maximum

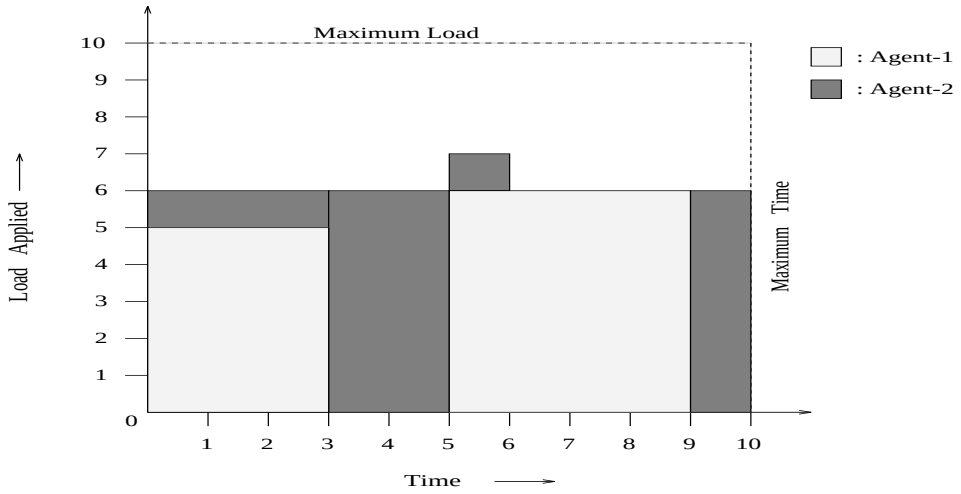

Figure 9: A resource sharing problem.

utility. The problem is that the second agent have no direct information about the load distribution on the system. This is a typical situation in reinforcement learning problems where the agent has to choose its actions based only on scalar feedback. This formulation is different from other load balancing problems used in MAL literature [41] where more short-term feedback is available to agents.

A single trial consists of an episode of applying loads until T time steps is completed or until the agent has exhausted its load-hours, whichever occurs earlier. Thus, through consecutive such trials the agent learns to distribute its load on the system in an optimal fashion. Figure 9 presents the load distribution used by the first agent as well as one of several optimal load distributions for the second agent in the particular problem we have used for experiments ($T = 10$ in this problem).

The Resource sharing problem is another example of a *tightly-coupled* system with agent actions interacting with each other at every time step. This domain can be considered to be *non-cooperative* because agents are not doing a common task, each of the agents need to coordinate with the others to achieve a utility that is most beneficial to itself.

Since the resource sharing problem produces rewards only after a series of actions are performed we used the PSP method of payoff distribution with the classifier system. Though BBA can also be used for payoff distribution in this problem, our initial experiments showed that PSP performed much better than BBA on this problem. The parameter values are

$\beta = 0.85$, $\gamma = 0.9$ for Q-learning and $\alpha = 0.1$ for PSP.

In this set of experiments the fitness-proportionate PSP did not converge even after 150,000 trials. Experimental results comparing Q-learning and semi-random PSP, PSP(SR), based classifier systems on the resource sharing problem is displayed in Figure 10. Results are averaged over 50 runs of both systems. Though both methods find the optimal load distribution in some of the runs, more often than not they settle for a less than optimal, but reasonably good distribution. PSP takes about twice as long to converge but produces a better load distribution on the average. The difference in performance is found to be significant at the 99% confidence level using a two-sample t-procedure. We believe this happens because all the active rules directly share the feedback at the end of a trial in PSP. In Q-learning, however, the external feedback is passed back to policy elements used early in a trial over successive trials. Interference with different action sequence sharing the same policy element (state-action pair) can produce convergence to sub-optimal solutions.

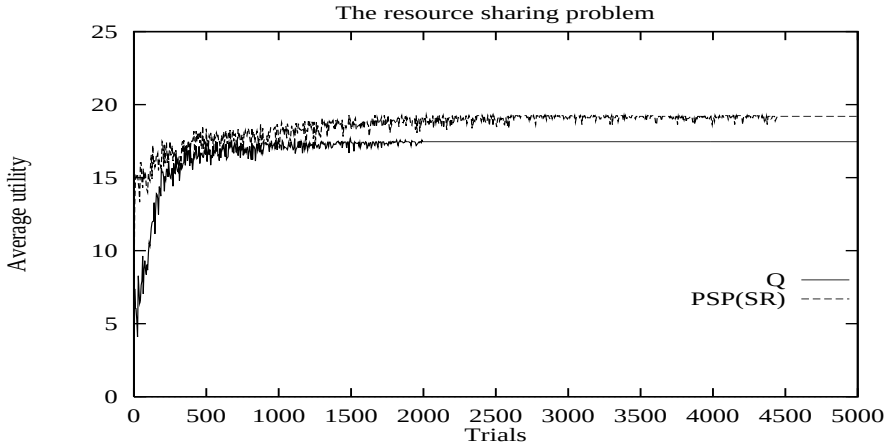

Figure 10: Comparison of PSP and Q-learning on the resource sharing problem.

Typical solutions produced by PSP and Q-learning differed in one important characteristic. PSP solutions will save some of its load for the last empty time-slot, whereas Q-learning solutions use up all the available load before that. Since PSP is able to utilize the last empty time slot on the channel, it produces better utility than Q-learning.

The above results show two things: 1) an agent can effectively use a classifier system to coordinate its actions effectively with no knowledge about the actions of the other agent

using the common resource, 2) semi-random action choice mechanism can be a more effective method for classifier systems than the commonly used fitness-proportionate action choice scheme.

We ran a further set of experiments in this domain where both agents were learning concurrently. In this mode, each agent would submit a load distribution to the system, and the submitted load distributions will be used to give them feedback on their utilities. Neither of the learning techniques was able to generate effective coordination between the agents. This set of experiments, alone, glaringly exposes the limitations of individual learning. It seems that in tightly coupled domains with delayed feedbacks it would be extremely unlikely that good coordination will evolve between agents learning at the same time. This is particularly true if there is one or very few optimal behavior pairings.

8 Conclusions and Future Research

In this paper we have addressed the problem of developing multiagent coordination strategies with minimal domain knowledge and information sharing between agents. We have compared classifier system based methods and Q-learning algorithms, two reinforcement learning paradigms, to investigate a resource sharing and a robot navigation problem. Our experiments show that the classifier based methods perform very competitively with the Q-learning algorithm, and are able to generate good solutions to both problems. PSP works well on the resource sharing problem, where an agent is trying to adapt to a fixed strategy used by another agent, and when environmental feedback is received infrequently. Results are particularly encouraging for the robot navigation domain, where all agents are learning simultaneously. A classifier system with the BBA payoff distribution allows agents to coordinate their movements with others without deviating significantly from the optimal path from their start to goal locations.

Experiments conducted on the block-pushing task demonstrate that two agents can coordinate to solve a problem better, even without developing a model of each other, than what they can do alone. We have developed and experimentally verified theoretical predictions of the effects of a particular system parameter, the learning rate, on system convergence. Other

experiments show the utility of using knowledge, acquired from learning in one situation, in other similar situations. Additionally, we have demonstrated that agents coordinate by learning complimentary, rather than identical, problem-solving knowledge.

Using reinforcement learning schemes, we have shown that agents can learn to achieve their goals in both cooperative and adversarial domains. Neither prior knowledge about domain characteristics nor explicit models about capabilities of other agents are required. This provides a novel paradigm for multi-agent systems through which both friends and foes can concurrently acquire coordination knowledge.

A particular limitation of the proposed approach that we have identified is the inability of individual, concurrent learning to develop effective coordination when agent actions are strongly coupled, feedback is delayed, and there is one or a few optimal behavior combinations. A possible partial fix to this problem would be to do some sort of staggered or lock-step learning. In this mode of learning, each agent can learn for sometime, then execute its current policy without modification for some time, then switch back to learning, etc. Two agents can synchronize their behavior so that one is learning while the other is following a fixed policy and vice versa. Even if perfect synchronization is not feasible, the staggered learning mode is likely to be more effective than the concurrent learning mode we have used in this paper.

A drawback of using reinforcement learning to generate coordination policies is that it requires considerable amount of data, and as such can only be used in domains where agents repeatedly perform similar tasks. Other learning algorithms with less data requirements can possibly be explored to overcome this limitation.

This paper demonstrates that classifier systems can be used effectively to achieve near-optimal solutions more quickly than Q-learning, as illustrated by the experiments conducted in the robot navigation task. If we enforce a more rigid convergence criteria, classifier systems achieve a better solution than Q-learning through a larger number of trials, as illustrated by the results obtained on the resource sharing domain. We believe, however, that either Q-learning or the classifier system can produce better results in a given domain. Identifying the distinguishing features of domains which allow one of these schemes to perform better will be a focus of our future research.

We have also shown that a semi-random choice of actions can be much more productive than the commonly used fitness-proportionate choice of actions with the PSP payoff distribution mechanism. We plan to compare the BBA mechanism with these two methods of payoff distribution.

We would also like to investigate the effects of problem complexity on the number of trials taken for convergence. On the robot navigation domain, for example, we would like to vary both the size of the grid as well as the number of agents moving on the grid to find out the effects on solution quality and convergence time.

Other planned experiments include using world models within classifier systems [3] and combining features of BBA and PSP [20] that would be useful for learning multiagent coordination strategies.

Acknowledgments

This research has been sponsored, in part, by the National Science Foundation under a Research Initiation Award IRI-9410180 and a CAREER award IRI-9702672.

References

[1] Andrew B. Barto, Richard S. Sutton, and Chris Watkins. Sequential decision problems and neural networks. In *Proceedings of 1989 Conference on Neural Information Processing*, 1989.

[2] Alan H. Bond and Les Gasser. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[3] Lashon B. Booker. Classifier systems that learn internal world models. *Machine Learning*, 3:161–192, 1988.

[4] L.B. Booker, D.E. Goldberg, and J.H. Holland. Classifier systems and genetic algorithms. *Artificial Intelligence*, 40:235–282, 1989.

[5] P. Brazdil, M. Gams, S. Sian, L. Torgo, and W. van de Velde. Learning in distributed systems and multi-agent environments. In *European Working Session on Learning*, Lecture Notes in AI, 482, Berlin, March 1991. Springer Verlag.

[6] Hung H. Bui, Dorota Kieronska, and Svetha Venkatesh. Negotiating agents that learn about others' preferences. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 114–119, Menlo Park, CA, 1996. AAAI Press.

[7] David Carmel and Shaul Markovitch. Incorporating opponent models into adversary search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 62–67, Menlo Park, CA, 1996. AAAI Press.

[8] Philip R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3):177–212, 1979.

[9] Susan E. Conry, Robert A. Meyer, and Victor R. Lesser. Multistage negotiation in distributed planning. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 367–384. Morgan Kaufman, 1988.

[10] Marco Dorigo and Hugues Bersini. A comparison of Q-learning and classifier systems. In *Proceedings of From Animals to Animats, Third International Conference on Simulation of Adaptive Behavior*, 1994.

[11] Edmund H. Durfee and Victor R. Lesser. Partial global planning: A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5), September 1991. (Special Issue on Distributed Sensor Networks).

[12] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, C-36(11):1275–1291, November 1987. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 268–284, Morgan Kaufmann, 1988.).

[13] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1):63–83, March 1989.

[14] Edmund H. Durfee and Thomas A. Montgomery. Coordination as distributed search in a hierarchical behavior space. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6):1363–1378, November/December 1991.

[15] Mark S. Fox. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(1):70–80, January 1981. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 140–150, Morgan Kaufmann, 1988.).

[16] Andrew Garland and Richard Alterman. Multiagent learning through collective memory. In Sandip Sen, editor, *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pages 33–38, Stanford University, CA, March 1996.

[17] Les Gasser and Michael N. Huhns, editors. *Distributed Artificial Intelligence*, volume 2 of *Research Notes in Artificial Intelligence*. Pitman, 1989.

[18] M.R. Genesereth, M.L. Ginsberg, and J.S. Rosenschein. Cooperation without communications. In *Proceedings of the National Conference on Artificial Intelligence*, pages 51–57, Philadelphia, Pennsylvania, 1986.

[19] Piotr J. Gmytrasiewicz, Edmund H. Durfee, and David K. Wehe. A decision-theoretic approach to coordinating multiagent interactions. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 62–68, August 1991.

[20] John Grefenstette. Credit assignment in rule discovery systems. *Machine Learning*, 3(2/3):225–246, 1988.

[21] Pan Gu and Anthony B. Maddox. A framework for distributed reinforcement learning. In Gerhard Weiß and Sandip Sen, editors, *Adaptation and Learning in Multi–Agent*

Systems, Lecture Notes in Artificial Intelligence, pages 97–112. Springer Verlag, Berlin, 1996.

[22] Joseph Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990. A preliminary version appeared in *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, 1984.

[23] Thomas Haynes and Sandip Sen. Learning cases to compliment rules for conflict resolution in multiagent systems. International Journal of Human-Computer Studies (to appear).

[24] John H. Holland. *Adpatation in natural and artificial systems.* University of Michigan Press, Ann Arbor, MI, 1975.

[25] John H. Holland. Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, J.G. Carbonell, and T. M. Mitchell, editors, *Machine Learning, an artificial intelligence approach: Volume II.* Morgan Kaufmann, Los Alamos, CA, 1986.

[26] Michael Huhns, editor. *Distributed Artificial Intelligence.* Morgan Kaufmann, 1987.

[27] L.P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of AI Research*, 4:237–285, 1996.

[28] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated reannealing. *Science*, 220:671–680, 1983.

[29] Victor R. Lesser. Multiagent systems: An emerging subdiscipline of AI. *ACM Computing Surveys*, 27(3):340–342, September 1995.

[30] 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.

[31] Sridhar Mahadevan. To discount or not to discount in reinforcement learning: A case study comparing R learning and Q learning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 205–211, 1993.

[32] Thomas W. Malone. Modeling coordination in organizations and markets. *Management Science*, 33(10):1317–1332, 1987. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 151–158, Morgan Kaufmann, 1988.).

[33] James G. March and Herbert A. Simon. *Organizations*. John Wiley & Sons, 1958.

[34] Maja J. Mataric. Learning in multi-robot systems. In Gerhard Weiß and Sandip Sen, editors, *Adaptation and Learning in Multi–Agent Systems*, Lecture Notes in Artificial Intelligence, pages 152–163. Springer Verlag, Berlin, 1996.

[35] K. Narendra and M.A.L. Thatachar. *Learning Automata: An Introduction*. Prentice Hall, 1989.

[36] Lynne E. Parker. Adaptive action selection for cooperative robot teams. In J. Meyer, H. Roitblat, and S. Wilson, editors, *Proc. of the Second International Conference on Simulation of Adaptive Behavior*, pages 442–450, Cambridge, MA, 1992. MIT Press.

[37] M V Nagendra Prasad, Susan E. Lander, and Victor R. Lesser. Cooperative learning over composite search spaces: Experiences with a multi-agent design system. In *Proceedings of Thirteenth National Conference on Artificial Intelligence*, pages 68–73, August 1996.

[38] Foster John Provost and Daniel N. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 74–79, Menlo Park, CA, 1996. AAAI Press.

[39] Jeffery S. Rosenschein. Synchronization of multi-agent plans. In *Proceedings of the National Conference on Artificial Intelligence*, pages 115–119, Pittsburgh, Pennsylvania, August 1982. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 187–191, Morgan Kaufmann, 1988.).

[40] Tuomas W. Sandholm and Robert H. Crites. On multiagent Q-learning in a semi-competitive domain. In Gerhard Weiß and Sandip Sen, editors, *Adaptation and Learning in Multi–Agent Systems*, Lecture Notes in Artificial Intelligence, pages 191–205. Springer Verlag, Berlin, 1996.

[41] A. Schaerf, Y. Shoham, and M. Tennenholtz. Adaptive load balancing: A study in multiagent learning. *Journal of Artificial Intelligence Research*, 2:475–500, 1995.

[42] Jurgen Schmidhuber. A general method for multi-agent reinforcement learning in unrestricted environments. In Sandip Sen, editor, *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, pages 84–87, Stanford University, CA, March 1996.

[43] Mahendra Sekaran and Sandip Sen. Learning with friends and foes. In *Sixteenth Annual Conference of the Cognitive Science Society*, pages 800–805, 1994.

[44] Sandip Sen. IJCAI-95 workshop on adaptation and learning in multiagent systems. *AI Magazine*, 17(1):87–89, Spring 1996.

[45] Sandip Sen, Mahendra Sekaran, and John Hale. Learning to coordinate without sharing information. In *National Conference on Artificial Intelligence*, pages 426–431, 1994.

[46] Yoav Shoham and Moshe Tennenholtz. On the synthesis of useful social laws for artificial agent societies (preliminary report). In *Proceedings of the National Conference on Artificial Intelligence*, pages 276–281, San Jose, California, July 1992.

[47] S. Sian. Adaptation based on cooperative learning in multi-agent systems. In Y. Demazeau and J.-P. Müller, editors, *Decentralized AI*, volume 2, pages 257–272. Elsevier Science Publications, 1991.

[48] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[49] Richard S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts at Amherst, 1984.

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

[51] C.J.C.H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge University, 1989.

[52] Gerhard Weiß. Learning to coordinate actions in multi-agent systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 311–316, August 1993.

[53] Gerhard Weiß, editor. *Distributed Artificial Intelligence Meets Machine Learning: Learning in Multi-Agent Environments*. Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, 1997.

[54] Gerhard Weiß and Sandip Sen, editors. *Adaptation and Learning in Multi–Agent Systems*. Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, 1996.

[55] Eric Werner. Cooperating agents: A unified theory of communication and social structure. In Les Gasser and Michael N. Huhns, editors, *Distributed Artificial Intelligence*, volume 2 of *Research Notes in Artificial Intelligence*, pages 3–36. Pitman, 1989.

[56] Holly Yanco and Lynn Andrea Stein. An adaptive communication protocol for cooperating mobile robots. In Stewart Wilson, editor, *From Animals to Animats: Proc. of the Second International Conference on the Simulation of Adaptive Behavior*, pages 478–485, Cambridge, MA, 1993. MIT Press.

[57] M. Yokoo, E. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the Twelfth International Conference on Distributed Computing Systems*, pages 614–621, 1992.

	Cooperative relationships		Non-cooperative relationships
	Individualistic learning	Shared learning	
Communicating agents			
Non-communicating agents			

Table 1: A categorization of multiagent domains for identifying learning opportunities.

F_A	F_B	Trials taken to reach Goal of Agent A
10	1	81
10	2	111
10	3	115
10	4	197
10	5	268

Table 2: Trials for agent A to reach its goal acting against agent B trying to reach its own goal (maximum forces applied by agents A and B are F_A and F_B respectively).

[58] Gilad Zlotkin and Jeffrey S. Rosenschein. Negotiation and conflict resolution in non-cooperative domains. In *Proceedings of the National Conference on Artificial Intelligence*, pages 100–105, July 1990.