Mutually Supervised Learning in Multiagent Systems

Claudia V. Goldman and Jeffrey S. Rosenschein
Computer Science Department
Hebrew University
Givat Ram, Jerusalem, Israel
ph: 011-972-2-658-5353
fax: 011-972-2-658-5439
email: clag@cs.huji.ac.il, jeff@cs.huji.ac.il

Abstract

Learning in a multiagent environment can help agents improve their performance. Agents, in meeting with others, can learn about the partner's knowledge and strategic behavior. Agents that operate in dynamic environments could react to unexpected events by generalizing what they have learned during a training stage.

In this paper, we propose several learning rules for agents in a multiagent environment. Each agent acts as the teacher of its partner. The agents are trained by receiving examples from a sample space; they then go through a generalization step during which they have to apply the concept they have learned from their instructor.

Agents that learn from each other can sometimes avoid repeatedly coordinating their actions from scratch for similar problems. They will sometimes be able to avoid communication at run-time, by using learned coordination concepts.

Subtopic: Learning in multiagent systems, Coordination

1 Introduction

Distributed Artificial Intelligence (DAI) is concerned with effective agent interactions, and the mechanisms by which these interactions can be achieved. One of the central issues in multiagent environments is that of appropriate coordination techniques. Much DAI research deals with this issue by giving pre-computed solutions to specific problems. For every new problem, the agents will start from scratch and derive the appropriate solution (even if it is an interaction instance identical to one they have seen before). For example, researchers have considered negotiation as a technique for deriving agreements that determine agent actions [Smith, 1978; Durfee, 1988; Kraus and Wilkenfeld, 1991; Zlotkin and Rosenschein, 1993]. These negotiation techniques invariably focus on a single encounter or set of encounters; agents do not (for example) improve their negotiation performance based on experience. Other DAI researchers have focused more on direct modeling of agents' beliefs and desires, as another way for an agent to decide what action to perform when dealing with others [Grosz and Kraus, 1993]. Again, learning rarely enters into this research; while the exploitation of a model of the opponent is studied, the actual derivation of the model rarely is.

Multiagent reactive systems have also been analyzed within DAI, where solutions are arrived at dynamically by reactive agents in multiagent environments. Social laws [Tennenholtz and Moses, 1989; Shoham and Tennenholtz, 1992] and cooperative state-changing rules [Goldman and Rosenschein, 1994] have been studied; these conventions give the agents a framework within which to act, to more harmoniously interact with the other agents participating in the same world. Learning has been investigated within this framework, particularly in [Shoham and Tennenholtz, 1994], which investigated how conventions can evolve when the Highest Cumulative Reward update rule is used (i.e., agents choose to perform the action that has yielded the highest payoff until then).

The advantages of having agents learn within a multiagent environment are clear. In Cooperative Problem Solving systems, cooperative behavior can be made more efficient when agents adapt to information about the environment and about their partners. In competitive Multiagent Systems, agents' performance within the environment can be improved if they can learn about the strategies and preferences of their opponents.

In this paper, we present a learning algorithm for a cooperative multiagent environment. The agents in our model first go through a training step, and are then able to choose their actions by generalizing what they have learned. The agents do not need to re-coordinate their actions for every new situation or problem. The main issue in our research is how to train the agents in a way that minimizes the number of mistakes in the generalization step.

This distributed learning approach to coordination is useful whenever agents do not have enough time to negotiate, when they exist in a dynamic environment and will benefit by adapting to unpredictable situations, and when the agents face similar problems repeatedly. The actual utility of having agents use learning instead of, for example, negotiation, will depend on the tradeoff between the time it takes to train an agent (which induces an upper bound on the probability of an agent making errors) and the time it would have taken him to re-coordinate his actions from scratch (along with the communication and computation overheads involved).

Section 2 presents an overview of related research that has dealt with the topic of multiagent learning, and presents the learning models of Probably Approximately Correct Learning and P-Concept Learning. In Section 3, we describe our specific scenario, the traffic signals domain. Afterwards, in Section 4 we explain the training and generalization steps in the learning process. Finally, in Section 5, we conclude and discuss several additional points for further investigation.

2 Background on Learning

2.1 Multiagent Learning

An example of a multiagent system in which every agent has a learning module is [Sian, 1991]. The agents' evaluate various hypotheses while having only partial knowledge about them. The agents can know more about a specific hypothesis by receiving messages from other agents expressing their level of confidence in that hypothesis. The general aim of the system is for agents to settle on the best hypothesis they can, given the knowledge they have.

The ACE and the ACG learning algorithms were proposed for a multiagent environment in [Weiß, 1993]. In this case, the agents can learn how to coordinate their actions by iteratively sending to one another bids for every action they can perform. These bids reflect the importance of the specific action relative to the agent's goal. The action that gets the highest bid is allowed to be executed. All incompatible actions are dropped, and the agents repeat the above steps while there exists an action with a non-zero bid proposed for it. The two algorithms differ primarily in the competition step; in the ACG algorithm, agents compete over a group of actions.

Reinforcement learning techniques have also been applied in multiagent scenarios. Agents that learn according to the Q-learning algorithm [Watkins and Dayan, 1992] and also cooperate with other agents by exchanging information (partial solutions, plans of action) can learn more quickly than agents that do not cooperate [Tan, 1993].

Littman proposed the use of Markov games as a framework for multiagent systems [Littman, 1994]. He focused on two-player games, where one agent plays the maximizer and the other the minimizer of the expected value $E\{\Sigma_{j=0}^{\infty}\gamma^j r_{i,t+j}\}$, where γ is a discount factor, and $r_{i,t+j}$ is the reinforcement value that agent i receives t steps after time j; he presented a minimax Q-learning algorithm for this problem.

Experiments using the Q-learning algorithm were also carried out in [Sen *et al.*, 1994] and in [Sandholm and Crites, submitted]. In [Sen *et al.*, 1994], two agents have to push a block to a goal position. Agents learned complementary policies without sharing any information with the other agent. The agents are actually not even aware of the existence of the other; they learn how to push the block by receiving reinforcement from the environment regarding the distance of the block from the optimal path. [Sandholm and Crites, submitted] covered an experimental study with the iterated prisoner's dilemma, where the players were Q-learning agents. In some experiments an agent played against the fixed TIT-FOR-TAT strategy, while in others it played against another Q-learning agent. The Q-learning algorithm has also been applied to packet routing [Littman and Boyan, 1993]. Each node in the network is a Q-learner.

Findler [Findler, 1991] has investigated how expert systems can control street traffic signals. In his domain there is only one processor for each intersection. This processor, the expert system, communicates with its four adjacent neighbors. The processors exchange information about the number and the speed of cars going in each of the four possible directions, features of the traffic's flow (type and rate of change), and expert advice. Based on this information, the experts at each intersection learn about the traffic pattern and can therefore choose which rules to apply.

2.2 Probably Approximately Correct Learning

In this section we present a brief overview of Probably Approximately Correct Learning (PAC Learning), which forms the basis of the technique given below (Section 2.3) for learning probabilistic concepts. Given the following concepts and notations [Anthony and Biggs, 1992], we can define the PAC learning algorithm.

- An *example space* $X \subseteq \Sigma^*$, where Σ is the alphabet for describing examples.
- A *concept space* C, $c \in C$ denotes a learned concept, i.e., $c : X \rightarrow \{0,1\}$
- A *sample of length* m is a sequence of m examples (i.e., $x = (x_1, x_2, \ldots, x_m) \in X$)
- A *training sample* $s = ((x_1, b_1), (x_2, b_2), \ldots, (x_m, b_m))$, $x_i \in X$ *and* $b_i \in \{0,1\}, b_i = c(x_i)$
- A *hypothesis space* $H \subseteq 2^X$
- The *error* of any hypothesis $h \in H$, with respect to a target concept t, is given by the probability of the event $h(x) \neq h(t)$
- μ denotes a probability distribution on X (i.e., the examples are chosen according to μ)
- $S(m,t)$ denotes the set of training samples of length m for a given target concept t, where the examples are taken from an example space X (i.e., if $x = (x_1, \ldots, x_m)$ *then* $s = ((x_1, t(x_1)), \ldots, (x_m, t(x_m)))$

Then, following [Valiant, 1984; Anthony and Biggs, 1992], we can state the following definition.

Definition 1 *The algorithm L is a probably approximately correct learning algorithm for the hypothesis space H if, given*

- *A real number δ ($0 < \delta < 1$);*
- *A real number ϵ ($0 < \epsilon < 1$);*

then there is a positive integer $m_0 = m_o(\delta, \epsilon)$ such that

- *For any target concept $t \in H$, and*
- *For any probability distribution μ on X;*

whenever $m \geq m_0$, $\mu^m\{s \in S(m,t) | er_\mu(L(s)) < \epsilon\} > 1 - \delta$.

2.3 Learning Probabilistic Concepts

Kearns and Shapire have proposed another learning model when the concept to be learned displays uncertain behavior [Kearns and Shapire, 1990]. This means that a probabilistic concept (p-concept) will be learned from examples that, although identical, will sometimes receive a positive label and sometimes a negative label. For example, if the concept to be learned is an event, which can be characterized by a parameter x, that occurs with probability $p(x)$, then the learner will receive training examples of the form (x_i,1) or (x_i,0), where the corresponding label is distributed according to $p(x)$. Formally, given X, the instance space, define the p-concept c to be a function, $c : X \rightarrow [0, 1]$. $c(x) = \Pr[x$ is a positive example of the p-concept $c]$. An oracle, the teacher, will choose a training example $x \in X$ according to the distribution of the objects in X. Then, it will label it 1 with probability $c(x)$ and 0 with probability $1 - c(x)$.

Following [Kearns and Shapire, 1990], a p-concept class over a domain X is learnable with a model of probability, if there is an algorithm A (the learner) such that for any target p-concept c, for any distribution D over X, for any inputs $\epsilon > 0$, $\delta > 0$, the algorithm A that has access to the oracle (the teacher), halts and with probability $1-\delta$ outputs a p-concept h that is an ϵ-good model of probability ($Pr_{x \in D}[|h(x) - c(x)| \leq \epsilon] \geq 1 - \epsilon$), for c with respect to D.

3 The Specific Scenario

In this paper, we consider how to apply learning techniques to the scenario of traffic signal controlling agents.[1] More specifically, we regard only one intersection with two agents, one responsible for the flow of traffic in each direction (no turns are allowed), as shown in Figure 1.

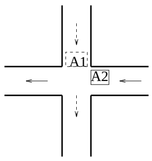

Figure 1: A Simple Scenario

Let us denote the vertical road r_1 and the agent (i.e., the traffic signal) that controls the traffic of this street A_1. Similarly, we'll name the horizontal road and its agent r_2 and A_2, respectively.

We will map real, continuous time into discrete units. We sample the traffic load for each discrete time unit, in each of the two crossing roads (i.e., how many cars arrive per hour). These real values are related to the Waiting-Times of the cars in the sampled road (that is, how long cars wait at the traffic signals).[2] These numbers induce a "waiting time distribution" over the hours of the day.

Each agent can choose to perform one of two possible actions, "red" or "green." If the agent performs red then the cars in the street that he is responsible for will have to wait, and if he performs green they can move in the appropriate direction. Each agent knows the sample values from the waiting time distribution of its cars. This distribution takes into consideration the different flow in traffic during different times of the day.

Although it is clearly preferable for the agents to coordinate their green/red signals so that traffic flows smoothly, it is also possible for both agents to show red (resulting in wasteful waiting) or for both agents to show green (resulting in traffic slowdown and potential collisions as cars navigate the under-regulated intersection).

Were the agents not to employ a learning algorithm, they might coordinate their actions by exchanging and comparing the current waiting times, and using some decision rule to decide which agent will be red and which will be green. The disadvantages would then be the cost of communication, the utter dependence on reliable communication to choose actions, and the fact that the decisions will be completely "local" in time, without regard for the histories of traffic flow (and while these considerations may or may not always be compelling for traffic light agents, they certainly play a role among other kinds of coordinating agents).

The agents' purpose is to learn which action they should perform, taking into account the existence of the other agent. The choice of which action to take (green or red) at any given time is based on the current information he has regarding his own stream of traffic, the sample values of his own waiting time distribution, and the information he has learned regarding the other agent's distribution.

Ideally, the two agents will choose those actions cooperatively (one is green while the other is red) so that the cars will wait as little as possible given the above intersection. The agents are inherently cooperative, and are not competing with one another. They are programmed to perform the learning step for the purpose of decreasing waiting time of cars, both for their own stream of traffic, and for the intersection as a whole (the perpendicular traffic stream). In this paper we do not explore any questions regarding the agents' motivations.

4 The Learning Algorithm

4.1 Learning in the Deterministic Case

The solution to this problem is divided into two steps. In the first step, each agent is trained by the second agent, that acts as his teacher. In the following notation, $i, j \in \{1, 2\}$. Agent A_i sends its list of sample values (i.e., a list of real values), that is a list of examples of waiting times of its cars to A_j. For each example, A_j will return

[1] The scenario is closely related to other job scheduling and information routing activities that automated agents might be involved in, for example, packet routing in a Wide Area Network. More generally, it is a simple example of a cooperative task for two agents, which can be applied to any situation where agents have to coordinate activity to be effective.

[2] The relationship of arrival density of cars to Waiting-Time depends on two other parameters, the "rate of removal" of cars from the intersection when the light is green (which might be assumed to be a constant), and the interval of time that the light is green, which is what we are trying to learn.

to A_i a pair $(example - value, action)$. The example-value runs over all the real values in the sample list sent by A_i to A_j, and the action is the action that A_j would want A_i to perform. The action can be red or green. A_j decides which label (i.e., action) to attach to the current example value according to its own sample values. Let us denote by $example - value_i^k$ the value of the example from A_i's list at line k. Then, A_j will decide upon the label red if $example_i^k \leq example_j^k$ and green otherwise. In other words, A_j would like A_i to switch to the red light each time A_j's cars are waiting more than A_i's.

We propose a learning rule that is based on each agent remembering the waiting times when his teacher wanted him to switch between the red and green lights. The agent has to remember only the w-t_i^k and w-t_i^{k+l}s where $a_i^k \neq a_i^{k+l}$ and $k+l$ is the first index with this characteristic.

For any new example taken from the same distribution used in the learning step, the agent is able to label it (i.e., he can decide upon which action to take). The rule for that will be as follows (see Figure 2).

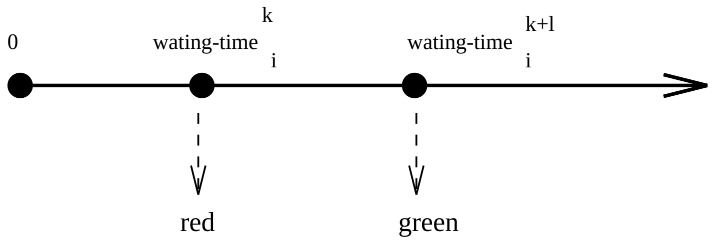

Figure 2: The Generalization Rule

Assume the new example is given by a value v (such that $WaitingTime_i^k \leq v \leq WaitingTime_i^{k+l}$), then agent A_i will perform red if

$$WaitingTime_i^k \leq v \leq \frac{WaitingTime_i^k + WaitingTime_i^{k+l}}{2}$$

and A_i will perform green if

$$\frac{WaitingTime_i^k + WaitingTime_i^{k+l}}{2} < v \leq WaitingTime_i^{k+l}.$$

We cannot analyze this learning rule according to the PAC learning model presented above because the concept the agents are learning is not deterministic as assumed by this model. In our learning rule, we assume that values that are close enough will be labeled similarly. This assumes indirectly that both agents' distributions do not change in an extreme way. If we want the agents to learn the other's distribution of values and then according to this distribution decide which action to perform, we have to look at a model that enables the agents to learn a non-deterministic concept. In the deterministic case, if the learner gives the same example-value twice then it is assumed that he will get the same label for both of them. Since the teacher labels the learner's training values according to his own values taken from his distribution, he might attach different labels for identical training values. Therefore, we propose another learning algorithm for the agents to learn a probabilistic concept, the distribution of the teacher's values.

4.2 Learning in the Probabilistic Case

Each agent will play the role of the learner and the teacher. The training step will not be exactly as in the PAC learning model. Now, the teacher will send to the learner all its training values (denote them by t_i). Then, whenever the learner faces a new waiting-value (denote it by v), he will count how many t_is are bigger than v and how many t_is are smaller than it. The learner will decide to perform red if there are more t_is that are bigger than smaller (similarly for the green action). The aim of this algorithm is that the number of t_is given by the teacher is enough (and minimal) for the learner to approximate the real weight (according to the teacher's distribution) of the t_i values. In this way, the learner tries to imitate the reaction of the teacher for the new value v. The learner will decide to perform red when he gets v at time k, when it is more likely that the teacher will get t at time k such that $t > v$.

Let us denote the teacher's distribution by T and his sample values by t_i. The teacher sends to the learner m sample values. Let's divide the real numbers into segments of size $\frac{\epsilon}{c_1}$. The actual value of each segment is given by the weight of the teacher values according to T. Since the t_is are only samples from T, the value of each segment can have an error. We will bound this error by $\frac{\epsilon}{c_2}$. Therefore, in each one of the segments there will be between $m \times (\frac{\epsilon}{c_1} - \frac{\epsilon}{c_2})$ to $m \times (\frac{\epsilon}{c_1} + \frac{\epsilon}{c_2})$ t_is. In addition, the learner can also err when he approximates the weight of the t_is that are larger (or smaller) than a given value, because the teacher might have more samples from a less dense portion of his distribution. Hence, for each middle point of each segment t_m, we will bound the number of t_is larger than t_m by $\frac{\epsilon}{c_3}$.

We want to find the sample size m such that the learner will err at most ϵ with confidence at least $1 - \delta$. Denote a segment by s. The three following inequalities have to be true:

$$\frac{\epsilon}{c_1} + \frac{\epsilon}{c_2} + \frac{\epsilon}{c_3} \leq \epsilon$$

$$\forall s_j Pr(\frac{|\{t_i \ s.t. \ (t_i > v) \wedge (t_i \in s_j)\}|}{m} - ActualWeight(s_j) > \frac{\epsilon}{c_2}) < \frac{\delta}{2}$$

$$Pr(\frac{|\{t_i \ s.t. \ t_i > v\}|}{m} - ActualWeight(all\ values > v\ according\ to\ T) > \frac{\epsilon}{c_3}) < \frac{\delta}{2}$$

From the first equation, we have that $c_1 = c_2 = c_3 = 3$. The left side of the second equation is bounded by $\left(2e^{-2m_1\frac{\epsilon^2}{9}}\right) * \frac{3}{\epsilon} < \frac{\delta}{2}$. Similarly, the left side of the third equation is bounded by $\left(2e^{-2m_2\frac{\epsilon^2}{9}}\right) * \frac{3}{\epsilon} < \frac{\delta}{2}$.

The sample size, m, that guarantees the learner to err at most ϵ with confidence 1-δ has a lower bound of

$max(m_1, m_2)$. For example, if we let the learner make a mistake of magnitude no greater than 1%, 95% of the time, he will need to see at least 907723 training values. If we let the learner err with magnitude at most 10%, 90% of the time, he will need to see at least 6382 training values.

4.3 Learning a P-Concept

To design autonomous agents, we'd like to let them interact with the minimal and sufficient amount of information needed for the achievement of their tasks. In the former case, each agent, acting as a teacher, needs to send to the learner agent all the values of the samples. We will show that even if the agents don't send each other these values, they can learn about the other's distribution as well. We will base our approach on the model presented in [Kearns and Shapire, 1990].

Each agent (A_i) will be trained by its teacher (A_j), as follows. Based on the model in [Kearns and Shapire, 1990], the oracle is represented by the teacher (i.e., A_j). The p-concept to be learned by both agents regards the distribution of the values of the teacher. More specifically, for our traffic problem, each agent wants to learn, for a given waiting time value v, what is the probability that the other agent will face a waiting time bigger or smaller than it. The oracle will get from the learner its sample values, x_i, and will return these values paired with a label $\in \{1, 0\}$, such that the labels have been generated according to the probability that most of the oracle's sample values, y_is, are larger than x_i. If we define the p-concept, $c(x)$, to be the number of y_is that are larger than x, then we've got that $c(x)$ is a non decreasing function. The value of $c(x)$ cannot decrease since it reflects a sum, namely how many $y_i\text{s} > x$. Following [Kearns and Shapire, 1990], the p-concept class of all non-decreasing functions $c : \mathcal{R} \rightarrow [0,1]$ is polynomially learnable with a model of probability.

In our case, mutual supervised learning, both agents will act also as an oracle. So, both agents will get the other's samples anyway, and hence they can learn which action to perform according to the algorithm presented in the former section. Nevertheless, we want to find a learning algorithm for a multiagent system by which the agents will learn how the teacher would like them to behave and at the same time the teacher is not revealing all the information he has about his sample values.

One solution is the use of a mediator, that will act as intermediate teacher. Each agent will send to the mediator its sample values (i.e., the waiting times). The mediator will play the role of two teachers, depending who his learner is. For example, when he acts as the teacher of A_i, he will send him A_i's sample values paired with a label that has been generated according to the data sent by A_j (similarly when A_j is the learner). Now, we can apply the p-concept learning model as presented in [Kearns and Shapire, 1990]. First each agent will send to the mediator its own values. Then, the learning algorithm presented there for the class of all non-decreasing functions from reals to [0,1] has to be applied twice, once when the oracle is the mediator and he sends the labels to A_i, and then again when the oracle sends the labels to A_j. Given a new waiting time value, v, to any of the agents, they could evaluate what is the probability of having most of the other agent's values larger than v. If this probability is greater or equal to 1/2 then the agent will perform red, otherwise it will perform green.

5 Summary and Future work

We have presented several learning rules for a multiagent environment. The probabilistic learning cases are presented schematically. In the first case (Figure 3) both agents send to each other all their corresponding sample values. X^i stands for all the sample values of agent A_i, and Y^j for all the sample values of agent A_j.

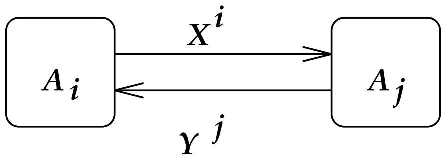

Figure 3: All Samples Case

In the second case (Figure 4) one agent (the learner) sends to its teacher all its sample values. The teacher sends him back its sample values paired with a label, generated according to the teacher's values. When A_j is the teacher, then he knows about X^i after A_i has sent him these values (similarly when A_i is the teacher).

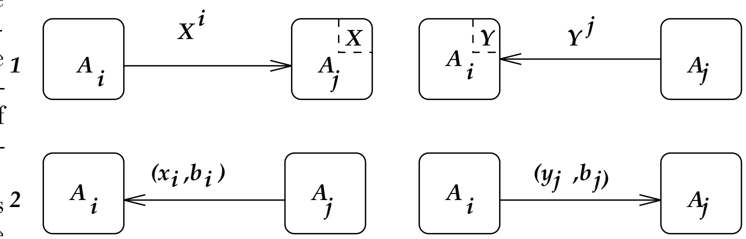

Figure 4: Learning with Labels

In the third case (Figure 5), each agent learns how to act with the other by receiving labeled examples sent by a mediator. The mediator knows both agents' sample values. The mediator sends to each agent the agent's sample values paired with a label generated according to the other agent's values. Therefore, the agents learn about the other distribution of values by getting only a label and not the values themselves.

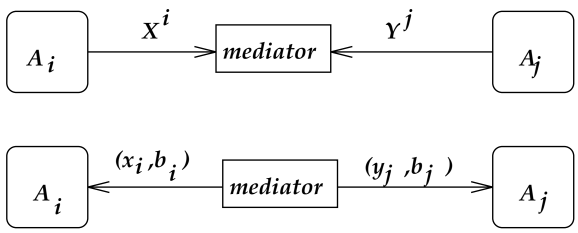

Figure 5: Learning with the Aid of a Mediator

Each agent plays the role of the learner and the teacher. The teacher instructs the learner how he should act, in order to please him. The learner generalizes the knowledge he had acquired by testing how close a new example is to the samples that he was trained with, relative to their average, or by evaluating the likelihood of how this new example would be labeled within the larger part of the sample values.

5.1 Future Work

In the traffic signal domain, we might also be interested in many agents, controllers of different roads, learning how to cooperate with one another. In this case, each agent will have many teachers, the agent's closest neighbors, who will send the agent labels for its sample values list. We can also consider another scenario, in which there are hierarchies among the streets (i.e., there are main roads, secondary roads, and so on).

In this paper we have assumed that the agents are benevolent. Alternative scenarios might involve selfish agents who could gain by lying. For example, the traffic signal domain is similar to a communication network domain, where each agent is responsible for messages flowing through the network. Agents might have an interest in delaying others' packets and letting their own flow. The agents can lie to each other by sending incorrect distributions or labels. These competitive learning issues are topics for future research, in which we hope to develop non-manipulable learning algorithms. Those algorithms will make it impossible for an agent to gain more by lying, whether as a learner or as a teacher.

Acknowledgements

We'd like to thank Dana Ron for valuable discussions on learning theory.

References

[Anthony and Biggs, 1992] Martin Anthony and Norman Biggs. *Computational Learning Theory.* Cambridge University Press, 1992.

[Durfee, 1988] Edmund H. Durfee. *Coordination of Distributed Problem Solvers.* Kluwer Academic Publishers, Boston, 1988.

[Findler, 1991] N.V. Findler. Distributed control of collaborating and learning expert systems for street traffic signals. In Lewis and Stephanon, editors, *IFAC Distributed Intelligence Systems*, pages 125—130. Pergamon Press, 1991.

[Goldman and Rosenschein, 1994] C. Goldman and J. Rosenschein. Emergent coordination through the use of cooperative state-changing rules. In *AAAI94*, pages 408–413, 1994.

[Grosz and Kraus, 1993] B. Grosz and S. Kraus. Collaborative plans for group activities. In *IJCAI93*, pages 367–373, Chambery, France, 1993.

[Kearns and Shapire, 1990] Michael J. Kearns and Robert E. Shapire. Efficient distribution-free learning of probabilistic concepts. In *Proceedings of the 31st Anual Symposium on Foundations of Computer Science*, pages 382–391, 1990.

[Kraus and Wilkenfeld, 1991] S. Kraus and J. Wilkenfeld. Negotiations over time in a multi agent environment: Preliminary report. In *IJCAI91*, pages 56–61, Sydney, Australia, August 1991.

[Littman and Boyan, 1993] M. Littman and J. Boyan. A distributed reinforcement learning scheme for network routing. technical report CMU-CS-93-165, Carnegie Mellon University, 1993.

[Littman, 1994] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning 1994*, pages 157—163, 1994.

[Sandholm and Crites, submitted] Tuomas W. Sandholm and Robert H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems Journal Special Issue on the Prisoner's Dilemma*, submitted.

[Sen *et al.*, 1994] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *AAAI94*, pages 426–431, 1994.

[Shoham and Tennenholtz, 1992] Y. Shoham and M. Tennenholtz. Emergent conventions in multi-agent systems: initial experimental results and observations (preliminary report). In *KR92*, Cambridge, Massachusetts, October 1992.

[Shoham and Tennenholtz, 1994] Y. Shoham and M. Tennenholtz. Co-learning and the evolution of social activity. Technical Report STAN-CS-TR-94-1511, Stanford Univ., 1994.

[Sian, 1991] S. Sian. Adaptation based cooperative learning multi-agent systems. In *Decentralized AI 2*, pages 257—272, 1991.

[Smith, 1978] Reid G. Smith. *A Framework for Problem Solving in a Distributed Processing Environment.* PhD thesis, Stanford University, 1978.

[Tan, 1993] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Machine Learning: Proceedings of the Tenth international conference*, pages 330—337, 1993.

[Tennenholtz and Moses, 1989] M. Tennenholtz and Y. Moses. On cooperation in a multi-entity model. In *IJCAI89*, pages 918–923, Detroit, Michigan, August 1989.

[Valiant, 1984] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134—1142, 1984.

[Watkins and Dayan, 1992] Christopher Watkins and Peter Dayan. Technical note Q-learning. *Machine Learning*, 8:279—292, 1992.

[Weiß, 1993] Gerhard Weiß. Learning to coordinate actions in multi-agent systems. In *IJCAI93*, Chambery, France, 1993.

[Zlotkin and Rosenschein, 1993] G. Zlotkin and J. S. Rosenschein. A domain theory for task oriented negotiation. In *IJCAI93*, pages 416–422, Chambery, France, 1993.