In *Proceedings of the Second International Conference on Autonomous Agents (Agents-98),*
Minneapolis, MN, USA, May 1998

Online Learning about Other Agents in a Dynamic Multiagent System

Junling Hu and Michael P. Wellman
Artificial Intelligence Laboratory
University of Michigan
Ann Arbor, MI 48109-2110, USA
{junling, wellman}@umich.edu
http://ai.eecs.umich.edu/people/{junling,wellman}

Abstract

We analyze the problem of learning about other agents in a class of dynamic multiagent systems, where performance of the primary agent depends on behavior of the others. We consider an online version of the problem, where agents must learn models of the others in the course of continual interactions. We implement various levels of recursive model in a simulated double auction market. Our experiments show that performance of an agent can be quite sensitive to its assumptions about the policies of other agents, and (not surprisingly), when there is substantial uncertainty about the level of sophistication of other agents, minimizing assumptions might be the best policy.

1 Introduction

A dynamic multiagent system, loosely speaking, is a system that changes over time due to the interaction of multiple agents. Examples of such systems abound; no doubt many will be represented at this conference. Popular demonstration applications include robotic soccer [1], the predator-prey pursuit game [5], and networked information systems [17]. Another example is a marketplace, where prices of goods continuously change as agents announce offers. As the rise of the Internet and electronic commerce continues, dynamic automated markets will be an increasingly important domain for software agents.

In designing agents to participate in an automated market, we face an array of general issues in dynamic multiagent systems. Agents need to consider not only the dynamic changes of the system, but also the reactions of other agents. Whereas agents may have some *a priori* basis for predicting the behavior of others, in typical situations much further evidence is revealed during the course of interaction. Thus, an agents' learning capacity can play a large role in its success. Depending on the type and quantity of information revealed through the interactions, an agent might learn about the others' joint or individual actions. In either case, this learning must be online because the other agents do not make themselves available for free trials.

In this paper, we adopt regression methods for online derivation of functional relations between other agents' actions and their internal states. The functional form of a regression model is based on an agent's assumptions of other agents' underlying behaviors. In this paper, we show that how an agent can form different models based on its different assumptions about other agents, and how such assumptions relate to the performance of the learning agent.

We define our problem in the context of a general class of dynamic multiagent systems. After the general discussion, we introduce a dynamic market trading game (interaction through continuous double auctions), which is an instance of the more general class of dynamic multiagent systems. We propose a series of regression-based learners for this domain, varying in strategy and depth of reasoning about others. In our experiments, learning agents with minimal assumptions about other agents tend to perform better than agents that attribute excess sophistication to their counterparts. However, when such attribution is warranted, agents can do better by learning the more sophisticated models.

2 Background and Related Work

We cast a dynamic multiagent system as a tuple, $\langle \mathcal{S}, \mathcal{A}, h, R \rangle$, where $\mathcal{S} = \prod_i \mathcal{S}^i$ is the joint state space, $\mathcal{A} = \prod_i \mathcal{A}^i$ the joint action space, h the state transition function, and $R = (R^1, \ldots, R^n)$ the vector of reward functions of the respective agents. At each time t, agent i receives its payoff r^i based on its local state s^i, its action a^i, and the actions of other agents a^{-i}. This can be characterized by a mapping: $(s^i, a^i, a^{-i}) \to r^i$. Whether the objective of agent i is to maximize the sum of all r^i's over time or just to maximize one period's r^i, agent i needs to correctly predict a^{-i}, the other agents' actions. This is where learning about other agents comes in. Since agents receive payoffs while they learn, their learning must also be online.

One can study online learning without explicitly modeling other agents. Research in this category has studied dynamic multiagent systems such as robotic soccer [1, 10] and the pursuit game [5]. The online learning method usually applied in such studies is reinforcement learning, of which Q-learning is the most popular. In most studies, Q-learning is used to derive the relationship $(s^i, a^i) \to r^i$. This ignores the impact of s^{-i} and a^{-i} on agent i's reward, r^i. De Jong [5] did, however, account for s^{-i} in his neural network model of the relationship $(s^i, s^{-i}) \to a^i$, where agents choose the best action based on current states. An explicit or implicit assumption of all these studies is that the learning agent need not worry about other agents' strategic response, either because other agents are reactive or because they follow a fixed rule.

One can also study online learning in single-state multiagent systems, which in most cases refer to repeated games. Such multiagent systems have no state transitions and hence are not dynamic systems in the strict sense. Online learning methods employed include fictitious play and Q-learning. Because the state is fixed, agents in a repeated game do not need to consider relations of actions to states. A theoretical framework is presented by Carmel and Markovitch [3] for learning in two-player repeated games. They propose that agents can learn about the other agent's action a^j by observing the history of joint actions $\{(a^i, a^j)\}$. The models to be learned are described by deterministic finite automata, which assume that the strategy of the other agent never changes and actions are discrete. Claus and Boutilier [4] apply Q-learning to a repeated coordination game. They study how agents can simultaneously learn about rewards coordinate strategies by judiciously combining exploitation and exploration.

Most studies of repeated games have considered two-player games. A few have admitted more than two players (e.g., the traffic light coordination game of Bazzan [2], and a stochastic social game investigated by Shoham and Tennenholtz [14]), but usually by having agents play two-player games with each other in a round-robin style. Schmidhuber and Zhao [12] consider a system with three self-interested agents. The agents use history data to choose actions that have been successful and remove actions that have failed. The success is measured by the accumulated reward at each point of evaluation.

Linear regression as an online learning methods has been adopted by Nadella and Sen [10] in designing soccer agents. A soccer agent learns about the linear relation between shooting error and shooting distance, and then decide to shoot or not based on the predicted error.

Explicit recursive modeling of other agents in multiagent settings was proposed by Gmytrasiewicz and Durfee [7]. Vidal and Durfee [17] implemented a recursive modeling approach in a particular exchange market. In further work, Vidal and Durfee [18] developed a more abstract frameworks of recursive modeling, for purposes of analyzing general multiagent learning behavior. In between the fully general abstraction and particular ad hoc models will be specifications of recursive modeling agents specialized to well-defined problem classes.

We proceed below to develop this specialization in two stages. We present first a general framework for dynamic multiagent systems and characterize recursive-modeling agents within this framework. Second, we map this characterization to a particular instance of dynamic multiagent systems, representing a prototypical dynamic market game.

3 Modeling Other Agents

3.1 Setup and assumptions

Our *dynamic multiagent system* consists of $n \geq 2$ agents, where each agent pursues its own objectives. The system starts at time 0, moving discretely forward to time T, or until the state satisfies some termination condition. Agents evaluate their outcomes based on the final state. (We could accommodate incremental-payoff models by accumulating the payoffs as part of the state.)

We assume all states are observable and each agent knows its own reward function. The only thing unknown is a^{-i}, the actions to be taken by other agents. The state evolves according to $s_{t+1} = h(s_t, a_t)$, where $s^i_{t+1} = h^i(s^i_t, a_t)$. That is, agent i's state at $t+1$ depends on the agent's current state and the current joint action. We assume that the transition function h is deterministic and stationary.

We allow both the state and action spaces be real (i.e., continuous) domains.

The agent's utility, U^i, is a function of its final state, s^i_T. Define i's *reward*, r^i, to be the improvement in utility at time t: $r^i_t \equiv U^i(s^i_t) - U^i(s^i_{t-1})$. Since

$$\sum_{t=1}^{T} r^i_t = \sum_{t=1}^{T} \left[U^i(s^i_t) - U^i(s^i_{t-1}) \right] = U^i(s_T) - U^i(s_0), \tag{1}$$

and $U^i(s_0)$ is a constant independent of i's actions, we see that maximizing final utility is equivalent to maximizing the sum of rewards.

However, even maximizing a single period's rewards,

$$r^i_{t+1} = U^i(h^i(s^i_t, a_t)) - U^i(s^i_t),$$

requires knowledge of the actions of other agents, a^{-i}_t. The object of the learning problem is to predict these actions—explicitly or implicitly—so that the agent can effectively choose its own.

To simplify the learning problem, we assume that the agent makes its decisions *myopically*, that is, considering only the current time period. To maximize the current reward at t (1), the agent solves

$$\arg\max_{a^i_t \in \mathcal{A}^i} U^i(h^i(s^i_t, a^i_t, a^{-i}_t)).$$

A generally applicable (albeit not optimal) approach is to form an estimate, $\hat{a}^{-i}_t$, of the other agents' actions, and solve the problem as if the estimate were correct. The entire decision then reduces to the question of how to form estimates.

3.2 Levels of modeling

Perhaps the most straightforward way to estimate a^{-i}_t is based on a time series analysis of the prior observations, $\{a^{-i}_\tau\}$, $\tau < t$. Agents taking this approach do not explicitly attempt to reason about what determines the a^{-i}. Adopting the terminology of Vidal and Durfee [18], we call such agents *0-level* agents, meaning that they do not model the underlying behavior of other agents. One example of a 0-level estimate is to take $\hat{a}^j_t$ as a linear function of prior observations,

$$\hat{a}_t{}^j = \sum_{k=1}^{t} w_k a^j_{t-k} \qquad (2)$$

A somewhat deeper way to learn a^{-i}_t is to learn, for each agent j, a model of the relationship between j's action, a^j_t, and its state, s^j_t. This amounts to learning a policy function for agent j. The policy function can be written as a function of j's state and j's estimate,

$$a^j_t = f^j(s^j_t, \hat{a}^{-j}_t).$$

An agent that attempts to model the policy functions (rather than just actions) of the others is called a *1-level* agent.

Note that our 1-level agent assumes that the policy function of agent j, f^j, is stationary. We call this the *fixed strategy* assumption. Although it is never true of learning agents, then decide to shoot or not based on the predicted error., it might be reasonable for an agent to adopt it when learning about others, as it greatly simplifies the modeling problem.

A natural extension is to consider agents that model others as 1-level. What this means is that the agent attempts to learn the policy function of other agents, taking into account that in this policy the subject agent is itself modeling the policies of others. Expressed mathematically, the *2-level* agent models the policy of agent j as

$$a^j_t = f^j(s^j_t, \{f^k(s^k_t, \hat{a}^{-k}_t) | k \neq j\}).$$

And of course one can define 3-level, ..., n-level agents by repeated deepening of this model.

3.3 Non-estimating agents

Although as one might expect, 0-level is as low as we can go, there are varying degrees of sophistication within levels—including the zeroth—that are worth distinguishing in a taxonomy of learning approaches. In particular, an agent might learn a policy without explicit consideration of a^{-i}_t at all, in which case its policy, $f^i(s^i_t)$, is a function of its local state alone.

An example of such an agent, introduced in our trading scenario below, is the *competitive* agent. To be competitive in a market context means to assume that one's own effect on the environment is negligible, in which case there is no advantage to speculating about the actions of others. This type of agent tends to behave "reactively", although this characterization is not as precisely defined as is "competitive" in the market context.

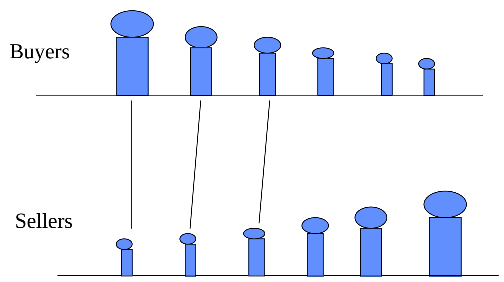
Our 1-level agent models the other agents as non-estimating 0-level types. To estimate the policy of agent j, it applies a locally weighted linear regression to the available history data, $\{(s^j_\tau, a^j_\tau) | \tau < t\}$. Given current state s^j_t, take its k nearest neighbors (defined by Euclidean distance), $\{s^j_{\tau_1}, \ldots, s^j_{\tau_k}\}$, and run a linear regression on the data points $\{(s^j_{\tau_1}, a^j_{\tau_1}), \ldots, (s^j_{\tau_k}, a^j_{\tau_k})\}$. This yields an estimation of the parameters α and β in the model

$$a^j = \alpha + \beta s^j.$$

The estimate of agent j's action will then be $\hat{a}^j_t = \alpha + \beta s^j_t$.

We can define a 2-level agent in a similar manner. The 2-level agent tries to model $f^j(s^j_t, \{f^k(s^k_t, \hat{a}^{-k}_t) | k \neq j\})$, where $f^k(s^k_t, \hat{a}^{-k}_t)$ is its model of agent j's model of agent k's policy. Since all agents have the same observations (except about their own policies), i's model of j's model of k (for $k \neq i$) is exactly what i's model of k would be if i were acting as a 1-level agent.

In our previous study [8], we investigated two types of agents: a competitive agent and a 1-level agent that modeled the others in the aggregate as non-estimating. Our learning model was also online, implemented in a market system called WALRAS [20]. Our experiments showed that when states are not observable, such incomplete information can lead the learning agent to a self-fulfilling suboptimal equilibrium. In this paper, we designed and tested four types of agent in a different, auction-based market system.



4 The Synchronous Double Auction Market

Auctions come in a wide range of types, distinguished by the way bidders submit their bids and how the allocations and prices are determined [9]. In a *double-sided* (or just *double*) auction, both buyers and sellers submit bids. A single agent may even submit both, offering to buy or sell depending on the price. Double auctions can be further classified according to such criteria as the timing of the bidding protocol. For example, in a *synchronous* double auction, all agents submit their bids in lockstep.

The book edited by Friedman and Rust [6] collects several studies of double auctions, including both simulations and game-theoretic analyses. In most simulations described, agents adopt simple bidding strategies without much modeling of other agents. Game-theoretic studies typically focus on equilibrium properties, tending to leave the process of reaching equilibrium unspecified.

4.1 Auction scenario

At the start of the scenario, each agent is endowed with designated amounts of m goods, each—except the last—associated with an auction. Each time period constitutes a bidding round for one auction, rotated in turn. Agents bid in an auction by posting buy and sell prices for one unit of the good. All prices are expressed in units of good m.

Given a set of bids, the auction matches the highest buyer to the lowest seller, if the buy price is greater than the sell price. The trading price for the match is the average of the two bid prices. Trades are implemented by transfer of one unit of the auctioned good for the designated amount of good m. The auction then matches the second highest buyer with the second lowest seller, and so on, until the highest remaining buyer is offering a lower price than the lowest remaining seller. At this point, the market proceeds in turn to the next auction. In the new round, agents post their buy and sell prices for the next good, and they are matched according to the prices they post. The market continues until no agents can be matched in any auction.

4.2 Framework

Agent i's local state at time t, s_t^i, is its holdings of the various goods, described by its endowment vector, $e_t^i = (e_1^i(t), \ldots, e_m^i(t))$. Agent i's action at time t, a_t^i, is its bid in the currently active auction, for good g. We describe the bid by a pair, $(P_{b,t}^i, P_{s,t}^i)$, denoting the buying and selling price offered for one unit of good g.

The reward for agent i at time t is given by

$$\begin{aligned} r_t^i &= U_{t+1}^i - U_t^i \\ &= U(e^i(t+1)) - U(e^i(t)) \end{aligned}$$

where

$$\begin{aligned} e_g^i(t+1) &= e_g^i(t) + z_t, \\ e_m^i(t+1) &= e_m^i(t) - P z_t, \text{ and} \\ e_l^i(t+1) &= e_l^i(t), \ l \neq g \text{ and } l \neq m. \end{aligned}$$

In these state update equations, P is the trading price, and $z_t \in \{-1, 0, 1\}$ is the quantity the agent trades at time t. When $z_t = 0$, the agent does not trade, and $r_t = 0$.

We assume that each agent i has a CES (Constant Elasticity of Substitution) utility function,

$$U(x) = \left(\sum_{g=1}^{m} \alpha_g x_g^{\rho} \right)^{\frac{1}{\rho}}, \tag{3}$$

where $x = (x_1, \ldots, x_m)$ is a vector of goods, the α_g are preference weights, and ρ is the substitution parameter. We choose the CES functional form for its convenience and generality—including quadratic, logarithmic, linear, and many other forms as special cases.

4.3 Reservation prices and bids

In constructing agent strategies, we dictate that they always choose actions leading to nonnegative payoffs. We can characterize this in terms of the agents' *reservation prices* [16]. The reservation price is defined as the maximum (minimum) price an agent is willing to pay (accept) for the good it wants to buy (sell). We can define agent i's buying and selling reservation prices, $\bar{P}_b$ and $\bar{P}_s$, as the prices such that its utility keeps constant when buying or selling one unit of good.

$$\begin{aligned} U(e_g - 1, e_m + \bar{P}_s, e_{-g,-m}) &= U(e_g, e_m, e_{-g,-m}), \quad (4) \\ U(e_g + 1, e_m - \bar{P}_b, e_{-g,-m}) &= U(e_g, e_m, e_{-g,-m}). \quad (5) \end{aligned}$$

It can be easily shown (proof omitted) that for quasi-concave utility (such as CES), the agent's reservation buy price is always lower than its reservation sell price.

Theorem 1 *For any strictly quasi-concave utility function, we always have $\bar{P}_b < \bar{P}_s$.*

Since the utility function is increasing in each of its components, for any trading price P such that $P \geq \bar{P}_s$ or $P \leq \bar{P}_b$, conducting the trade can never degrade the agent's utility. It follows then from Theorem 1 that an agent posting buy price P_b and sell price P_s such that its utility never decreases by trading at those prices will always satisfy $P_b \leq P_s$. From this we can show that an agent will never be both a buyer or seller in the same market.

Theorem 2 *Assuming an agent's sell bid exceeds its buy bid, the agent can never be matched both as buyer and a seller on the same market.*

Proof: Let the agent be i. Suppose agent i can be matched to agent j as a buyer, and matched to agent k as a seller. Then we have $P_b^i \geq P_s^j$ and $P_b^k \geq P_s^i$. From above, we have $P_s^j \leq P_b^i < P_s^i \leq P_b^k$. However, according to the matching rule, the highest buyer should be matched to the lowest seller, and so on. Thus, agent k should be matched to agent j instead of i, and we have a contradiction. □

4.4 Four types of agents

Our first type of agent, the *competitive*, does not attempt to model the other agents. Thus, it is a 0-level non-estimating agent, as described in Section 3.3. The competitive agent always chooses its reservation price $\bar{P}_b$ and $\bar{P}_s$. Since the trading price is the average of the buyer's and seller's bids, it is always at least $\bar{P}_s$ (when the competitive agent sells), or at most $\bar{P}_b$ (when the competitive agent buys).

The other three types of agents choose their bidding prices (P_b, P_s) based on their predictions of other agents. Although they differ in how they estimate other agents' actions, they are identical in acting according to their best response to this estimate.

4.4.1 Best-response bidding

Let $\{\hat{P}_b^1, \ldots, \hat{P}_b^n\}$ and $\{\hat{P}_s^1, \ldots, \hat{P}_s^n\}$ be the projected buying and selling prices of other agents. Let the agent we are interested in be i, with reservation prices $\bar{P}_b$ and $\bar{P}_s$. Suppose agent i can be matched as a buyer according to its reservation price, and is matched to seller j. By the matching criterion, we must have $\bar{P}_b \geq \hat{P}_s^j$, otherwise neither agent would want to trade. An agent should always choose its buy bid P_b such that $P_b \leq \bar{P}_b$, and indeed should bid as low as possible in order to maximize its payoff. However, if it bids lower than $\hat{P}_s^j$, it will lose the chance to trade with agent j.

Another way agent i might lose the opportunity to trade is if its buy price is lower than the other potential buyers. Suppose that agent k submits the highest buy bid among all potential buyers. If $\hat{P}_b^k \geq P_b$, agent k will get the match with agent j.

Therefore, according to its estimates of other agents' bids, agent i maximizes its reward by choosing

$$P_b = \max(\hat{P}_s^j, \hat{P}_b^k + \epsilon),$$

for ϵ a small positive constant representing the minimal bid increment. We can perform a similar simple analysis to determine the optimal buy and sell bid given all the other buy and sell bids.

If agent i predicts that it cannot be matched either as a buyer or a seller according to its reservation prices, it simply bids these reservation prices.

4.4.2 Three types of learning agents

A 0-level agent does not model the underlying policy functions of other agents. It models the actions of other agents by looking at the history data of those actions, and uses time series technique to predict the actions in next time period. For any other agent j, the 0-level agent predicts P_t^j by

$$P_t^j = P_{t-1}^j + \alpha(P_{t-1}^j - P_{t-2}^j). \tag{6}$$

A 1-level agent tries to estimate a fixed policy functions of other agents. It assumes that other agents are naive agents, which means other agent's actions are only function of their own states. Assuming the policy functions are linear, the estimation of P_t^j is

$$P_t^j = \alpha + \beta e_t^j.$$

A 2-level agent, like a 1-level agent, models the policy functions of other agents. But a 2-level agent assumes that other agents are learning at the same time. We adopt a simplified 2-level model: $P_t^j = f(e_t^j, e_t^{-j})$. We found that a linear regression method does not work well. One reason is the correlation between the independent variables e_t^j, e_t^{-j}. In addition, the high dimensionality of input data requires large amount of data in order to get unbiased estimation. Since we assume agents take only a small sample of history data (fixed window length), we need to adopt a nonparametric regression method. We use the K-Nearest Neighbor method in this paper. For current joint state $(e_t^1, \ldots, e_t^m)$, take its K nearest neighbors $\{(e_{t_1}^1, \ldots, e_{t_1}^m), \ldots, (e_{t_k}^1, \ldots, e_{t_k}^m)\}$ and the corresponding actions of agent j, $\{P_{t_1}^j, \ldots, P_{t_k}^j\}$. The estimation of P_t^j is:

$$P_t^j = \sum_{l=1}^{K} W_l P_{t_l}^j$$

where W_l is the weight of data point e_l and is defined as $W_l = \frac{\frac{1}{d_l}}{\sum_{l'=1}^{k} \frac{1}{d_{t_{l'}}}}$ where d_l is the distance between the data point $(e_l^1, \ldots, e_l^m)$ and the query point $(e_t^1, \ldots, e_t^m)$, $d_l = \sqrt{(e_l^1 - e_t^1)^2 + \cdots + (e_l^m - e_t^m)^2}$.

4.5 Convergence

By convergence, we mean that the system reaches a termination condition. In our system, termination is mandated when, for all goods, all the buy prices are lower than all the sell prices. That means no agents can be matched as a buyer or a seller, and no trading can happen.

Our auction model satisfies the conditions of an Edgeworth process [15], where the utility of agents never decrease through trade. In traditional study of Edgeworth processes, it is assumed that all agents are competitive. But in our system, agents can take strategic actions, which change the system behavior. Although we cannot rely on the traditional proof, our process shares the convergence property with competitive Edgeworth processes.

Theorem 3 *The synchronized double auction market converges.*

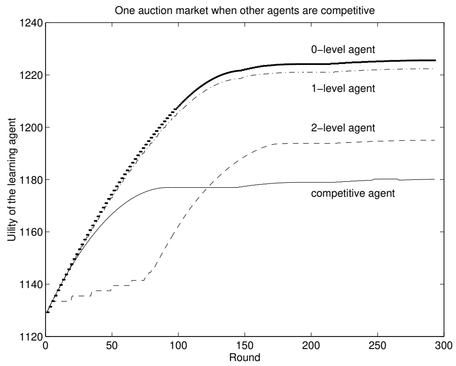
Proof: Let $U_t = \sum_{i=1}^{n} U^i(e_t^i)$ be the sum of utilities of all agents. Let $\bar{U} = \sum_{i=1}^{n} U^i(E)$, where E is the total endowment in the system. In a pure exchange system such as our auction market, the total endowment E remains constant. For each agent i, $e_t^i \leq E$. Thus $U_t \leq \bar{U}$ for any t, that is, U_t is bounded above by $\bar{U}$. Since we stipulate that agents always bid to get non-negative payoff, their utility will always be greater than or equal to that in last period. Thus the sum of all agents' utility, U_t, is nondecreasing. That is, $\frac{\partial U_t}{\partial t} \geq 0$. Since U_t is also bounded, the sequence $\{U_t\}$ will converge. □

5 Experiments

There are six agents and two types of goods in our experiments. Each agent starts with random endowment of both goods. Prices are in units of good 2; thus, all auction activities are for good 1.

Each agent has the same CES utility function as defined in (3). We let $\alpha_j = 1$ for all j, and $\rho = \frac{1}{2}$, and so the utility function becomes $U(x) = \left(\sum_j (x_j)^{\frac{1}{2}}\right)^2$. The reservation prices are calculated according to (4) and (5),

$$\bar{P}_b = 2((e_g(e_g+1))^{\frac{1}{2}} + (e_m(e_g+1))^{\frac{1}{2}} - (e_g e_m)^{\frac{1}{2}} - e_j) - 1,$$
$$\bar{P}_s = -2((e_g(e_g-1))^{\frac{1}{2}} + (e_m(e_g-1))^{\frac{1}{2}} - (e_g e_m)^{\frac{1}{2}} - e_g) - 1.$$



We test four types of agents: the competitive agent, 0-level agent, 1-level agent and 2-level agent. We put these agents in four kinds of environments where all other agents are: (1) competitive agents; (2) 0-level learning agents; (3) 1-level learning agents; (4) 2-level learning agents. Just to reiterate our definitions, a competitive agent does not use any information about other agents. It choose its action based on its individual optimization problem. A 0-level agent makes no assumption about other agents but use the data of other agents' actions. A 1-level agent assumes other agents are competitive. A 2-level agent assumes other agents are 1-level agents.

In each of the four environments, we randomly configure the initial endowment of all agents for 50 runs of the market. For each configuration, we change the type of the first agent for four times, and run the experiments for each type and compare the performance of the first agent for each type it assumes.

The figure above shows the experimental results in one configuration. Note that one cannot draw any conclusions from the magnitude of utility differences, since utility measures are only ordinally scaled.

Figure 1. Other agents are comp agents

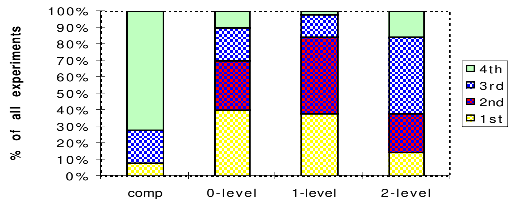


Figure 1 presents results for an environment where all other agents are competitive agents. For this environment, we predict that a 1-level agent will perform

Figure 2. Other agents are 0-level agents

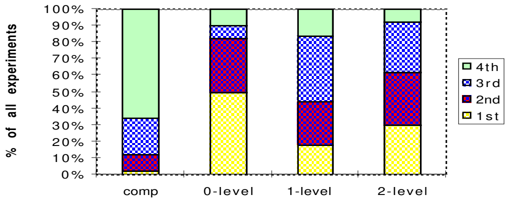


Figure 3. Other agents are 1-level agents

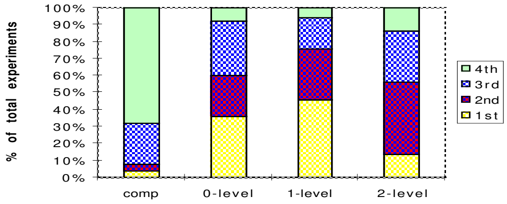
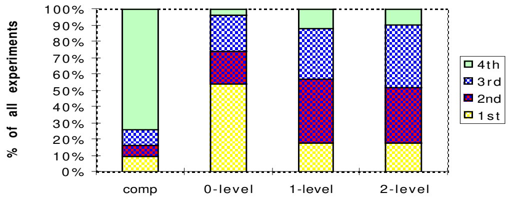


Figure 4. Performance when other agents are 2-level agents



the best because the it has correct assumption about other agents of their being competitive. As we can see, out of 88% of all experiments, the 1-level agent ranks among the top two types. The 0-level agent has similar performance only 70% of time. The 2-level agent competitive agents perform much worse than the other types.

Figure 2 presents results for an environment where all other agents are 0-level agents. For this environment, we would not expect 1-level or 2-level agents to perform better than the 0-level agent because these agents make incorrect assumptions about their counterparts. From Figure 2, we see that the 0-level agent, which achieves first- or second-best performance 82% of the time, is clearly the best among the types. The competitive agent still performs the worst.

When we change environments to one in which all other agents are 1-level, the experimental results (Figure 3) show that the 2-level agent performs worse than the 1-level agent, but almost as well as the 0-level agent. This is a little unexpected. We expected that the 2-level agent should do well since it correctly assumes other agents are 1-level agents. Our explanation is that the 2-level agent we implemented is a simplified version of a full 2-level agent, whose model should include all possible variables. Computationally, this is not feasible, and it remains an open questions how to form a computationally efficient and correct model of 2-level agent.

Figure 4 shows the results of an environment where all other agents are 2-level agents. In this environment, we did not expect either 1-level and 2-level agent to perform well, because these agents make incorrect assumptions about the other agents. The experimental results confirm our expectation. Both 1-level and 2-level agents perform worse than the 0-level agent, who achieves first- or second-best performance 73% of the time.

Our experiments suggest that, in most cases, the 0-level learning agent performs better than any the other types, especially when these others make incorrect assumptions about the environment. Our experiments also show that an agent with correct assumptions can do better than the agent with no assumptions, for example, the 1-level agent performs best when other agents are competitive. Finally, our experiments show that the competitive agent performs worst in all four cases. This suggests that if an agent does not use any information about other agents in the system, the agent will perform worse than those who do use the information.

Although the particulars differ, our results are consistent with recent studies of strategic versus competitive behavior in computational markets, conducted by ourselves and others [8, 11]. While behaving strategically can always potentially produce superior outcomes, it can also be far riskier than adopting a less aggressive, competitive approach in assumptions about other agents.

6 Conclusions

In this paper we analyze the issue of learning about other agents in a dynamic multiagent system with a continuous state-space. We test different types of learning models in our double auction market system. Our experimental results show that modeling other agents can be tricky. In general, an agent does not know what kind of strategies other agents adopt. An agent can learn about the general pattern of other agents' actions by observing the history data. But to learn the underlying model of those actions, an agent must make some assumptions about the form of the other agents' behaviors. If these assumptions are wrong, the agents may perform badly, regardless of the quality of their estimation methods.

References

[1] Tucker Balch. Learning roles: Behavioral diversity in robot teams. In Sen [13].

[2] Ann L. C. Bazzan. Evolution of coordination as a metaphor for learning in multi-agent systems. In Weiss [19]. Seleted papers from ICMAS'96 Workshop LIOME and ECAI'96 workshop LDAIS.

[3] David Carmel and Shaul Markovitch. Opponent modeling in multi-agent systems. Springer, 1996. Proceedings of IJCAI'95 Workshop.

[4] Caroline Claus and Boutilier Craig. The dynamics of reinforcement learning in cooperative multiagent systems. In Sen [13].

[5] Edwin De Jong. Non-random exploration bonuses for online reinforcement learning. In Sen [13].

[6] Daniel Friedman and John Rust, editors. *The Double Auction Market.* Addison-Wesley, 1993.

[7] Piotr J. Gmytrasiewicz and Edmund H. Durfee. A rigorous, operational formalization of recursive modeling. In *Proceedings of the First International Conference of Multiagent Systems.* AAAI Press, 1995.

[8] Junling Hu and Michael P. Wellman. Self-fulfilling bias in multiagent learning. In *Proceedings of the Second International Conference on Multiagent Systems.* AAAI Press, December 1996.

[9] R. Preston McAfee and John McMillan. Auctions and bidding. *Journal of Economic Literature*, 25:699–738, 1987.

[10] Rajami Nadella and Sandip Sen. Correlating internal parameters and external performance. In Weiss [19]. Seleted papers from ICMAS'96 Workshop LIOME and ECAI'96 workshop LDAIS.

[11] Tuomas Sandholm and Fredrik Ygge. On the gains and losses of speculation in equilibrium markets. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 632–638, Nagoya, Japan, 1997.

[12] Jurgen Schmidhuber and Jieyu Zhao. Multi-agent learning with the success-story algorithm. In Weiss [19]. Seleted papers from ICMAS'96 Workshop LIOME and ECAI'96 workshop LDAIS.

[13] Sandip Sen, editor. *Collected papers from the AAAI-97 workshop on multiagent learning.* AAAI, AAAI Press, 1997.

[14] Yoav Shoham and Moshe Tennenholtz. On the emergence of social conventions: Modeling, analysis, and simulations. *Artificial Intelligence*, 94(1-2):139–166, July 1997.

[15] Hirofumi Uzawa. On the stability of Edgeworth barter process. *International Economic Review*, 3(2):218–232, May 1962.

[16] Hal R. Varian. *Microeconomic Analysis.* W. W. Norton & Company, New York, third edition, 1992.

[17] José M. Vidal and Edmund H. Durfee. The impact of nested agent models in an information economy. In *Proceedings of the Second International Conference on Multiagent Systems.* AAAI Press, December 1996.

[18] José M. Vidal and Edmund H. Durfee. Agents learning about agents: A framework and analysis. In Sen [13].

[19] Gerhard Weiss, editor. *Distributed Artificial Intelligence Meets Machine Learning: learning in multiagent environment.* Springer, 1997. Seleted papers from ICMAS'96 Workshop LIOME and ECAI'96 workshop LDAIS.

[20] Michael P. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–22, 1993.