Dynamic Pricing Based on Asymmetric Multiagent Reinforcement Learning

Ville Könönen*
Neural Networks Research Centre, Helsinki University of Technology, P.O. Box 5400, FI-02015 HUT, Finland

A dynamic pricing problem is solved by using asymmetric multiagent reinforcement learning in this article. In the problem, there are two competing brokers that sell identical products to customers and compete on the basis of price. We model this dynamic pricing problem as a Markov game and solve it by using two different learning methods. The first method utilizes modified gradient descent in the parameter space of the value function approximator and the second method uses a direct gradient of the parameterized policy function. We present a brief literature survey of pricing models based on multiagent reinforcement learning, introduce the basic concepts of Markov games, and solve the problem by using proposed methods. © 2006 Wiley Periodicals, Inc.

1. INTRODUCTION

Reinforcement learning methods have recently been established as practical tools for solving Markov Decision Processes (MDPs). The main assumption behind these models is that the environment of the learning agent obeys the Markov property; that is, state transition probabilities depend only on the state of environment and the action selection of the learning agent. In multiagent settings, the Markov property does not hold anymore and this can lead to suboptimal results. One way to circumvent this problem is to use Markov games that are natural extensions of MDPs to multiagent settings.

The main aim of this article is to test asymmetric multiagent reinforcement learning methods with a dynamic pricing problem. In the problem, there are two competing brokers that sell identical products to customers and thus compete on the basis of price. This pricing problem was originally proposed by Tesauro and Kephart,[1] where the problem was solved by using a single-agent reinforcement learning method. The proposed method led to very good results in the cases where only one of the agents is learning and the other keeps its pricing strategy fixed. If both agents learn, the proposed method did not always converge. In this article, we model the problem as a Markov game and solve it by using two numerical

*e-mail: ville.kononen@hut.fi.

INTERNATIONAL JOURNAL OF INTELLIGENT SYSTEMS, VOL. 21, 73–98 (2006)
© 2006 Wiley Periodicals, Inc. Published online in Wiley InterScience (www.interscience.wiley.com). • DOI 10.1002/int.20121

learning techniques with asymmetric multiagent reinforcement learning. The proposed methods converged in every test case and the results were very promising.

The idea of using a heuristic approach to model foresight-based agent economies was originally proposed by Tesauro and Kephart.[2] The approach was then extended to utilize Q-learning in Ref. 1. When pricing applications contain a large number of possible prices, the state and action spaces become huge and lookup-table-based reinforcement learning methods become infeasible. To overcome this problem, the Q-function was approximated with different function approximators in Refs. 3 and 4.

Multiagent reinforcement learning methods have been discussed earlier by many authors. Existing methods can be roughly divided into three distinct groups: (1) methods utilizing direct gradients of agents' value functions, (2) methods that estimate the value functions and then use this estimate to compute an equilibrium of the process, and (3) methods that use direct policy gradients.

Early methods for multiagent reinforcement learning include, for example, Refs. 5 and 6. The method presented in Ref. 5 uses a simplified version of Q-learning to estimate agents' value functions. This method can fail to converge in some difficult coordination problems and some improvements aiming to overcome these problems were published in Refs. 7 and 8. Moreover, a set of performance comparisons between single-agent reinforcement learning and multiagent reinforcement learning were performed in Ref. 5. In Ref. 6, a simple gradient-based method is used to optimize agents' value functions directly so that it is always a best response to opponents' changing strategies. In all of these papers, the methods are tested with repeated games and deterministic reward values. In Ref. 9, Kapetanakis et al. propose the technique that converges almost always in fully stochastic environments, that is, when rewards are stochastic.

A more recent study falling into the third category is Ref. 10, which uses a policy gradient method originally proposed by Sutton et al.,[11] in multiagent context. The policy gradient method tries to find the optimal policy from a restricted class of parameterized policies. However, this method leans on the Markov property of the environment and is not thus directly suitable for multiagent domains. Bowling and Veloso solve this problem by using the WoLF principle[12] to adjust the learning rate so that the convergence is guaranteed (albeit only with very simple problems). Another study on policy gradients is Ref. 13, in which the VAPS framework originally proposed by Baird and Moore[14] for single-agent domains is expanded for cooperative games.

The first learning method for multistate Markov games was proposed by Littman.[15] He introduced a Q-learning method for Markov games with two players and a zero-sum payoff structure. This method is guaranteed to converge from arbitrary initial values to the optimal value functions. However, the zero-sum payoff structure can be a very restrictive requirement in some systems and thus Hu and Wellman extended this algorithm to general-sum Markov games.[16] Unfortunately, their method is guaranteed to converge only under very restrictive conditions. Littman proposed a new method,[17] which relaxes these limitations by adding some additional (a priori) information about the roles of the agents in the system. Wang and Sandholm proposed a method that is guaranteed to converge with any team

Markov game to the optimal Nash equilibrium.[18] Conitzer and Sandholm presented an algorithm that converges to a Nash equilibrium in self-play and learns to play optimally against stationary opponents.[19]

It still remains as an open question if there exist computationally efficient methods for computing Nash equilibria of finite games. To overcome this problem, Greenwald and Hall proposed a multiagent reinforcement learning method that uses the correlated equilibrium concept in place of the Nash equilibrium.[20] Correlated equilibrium points can be calculated using linear programming and thus the method remains tractable also with larger problem instances. Some complexity results about Nash equilibria can be found in Ref. 21.

A totally different approach to multiagent reinforcement learning is the collective intelligence (COIN) architecture proposed by Wolpert and Tumer.[22] The COIN architecture can been seen rather as a framework for adjusting single-agent reinforcement learning algorithms for multiagent domains than a stand-alone method for multiagent reinforcement learning. The main idea of COIN is to define reward signals for each agent according to some global fitness measure. Due to the overall structure of the COIN, the method is very scalable and remains robust as the problem size scales up. The COIN framework is also used in many large-scale real-world applications; see, for example, Refs. 23 and 24.

Our previous contributions in the field of multiagent reinforcement learning include an asymmetric multiagent reinforcement learning method,[25,26] which introduces an alternative solution concept to the Nash solution concept in Markov games, that is, the Stackelberg solution concept. Additionally, we have proposed a gradient-based learning method for both symmetric and asymmetric multiagent reinforcement learning[27] and a direct policy gradient method.[28,29] These papers are methodological and serve as stepping stones for this more application-oriented work.

We begin the article by introducing the background and basic solution concepts of game theory. Then we briefly go through the theory behind Markov decision processes and introduce some learning methods used with the multiagent reinforcement learning problem. In Section 5, we discuss numerical learning methods in Markov games. In Section 6, we introduce a pricing model of two brokers selling identical products and solve it by using proposed learning methods.

2. GAME THEORY

This section is mainly concerned with the basic problem settings and definitions of game theory. We start with some preliminary information about mathematical games and then proceed to their solution concepts, which are essential for the rest of the article. A thorough basic text on game theory is Ref. 30.

2.1. Basic Concepts

Mathematical games can be represented in different forms. The most important forms are the *extensive* form and the *strategic* form. Although the extensive

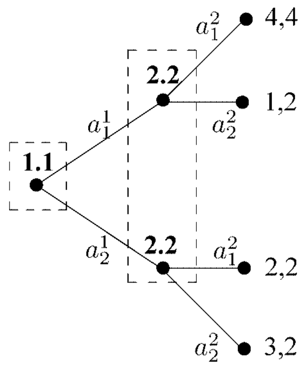

Figure 1. An example of a game in extensive form. Nodes 1.1 and 2.2 are decision nodes of players 1 and 2, respectively. Each arch connected to a decision node (marked with a) is denoting the decision of the corresponding player. Dashed boxes are information states for the corresponding player; for example, player 2 does not observe the actual strategy choice of player 1. The ith number in a leaf node is the resulting payoff for player i.

form is the most richly structured way to describe game situations, the strategic form is conceptually simpler and it can be derived from the extensive form. In this article, we use games in strategic form for making decisions at each time step.

Examples of both extensive form and strategic form games can be seen in Figures 1 and 2. Games in strategic form are usually referred to as *matrix games* and particularly in the case of two players, if the payoff matrices for both players are separated, as *bimatrix games*. In general, an N-person matrix game is defined as follows.

Definition 1. *A* matrix game *is a tuple* $\Gamma = (A^1, \ldots, A^N, r^1, \ldots, r^N)$*, where* N *is the number of players,* A^i *is the strategy space for player i, and* $r^i : A^1 \times A^2 \times \cdots \times A^N \to \mathbb{R}$ *is the payoff function for player i.*

In a matrix game, each player i simultaneously chooses a strategy $a^i \in A^i$. In addition to pure strategies A^i, we allow the possibility that the player uses a

	a_1^2	a_2^2
a_1^1	**2,2**	**4,0**
a_2^1	**1,0**	**3,1**

Figure 2. An example of a game in strategic form. The rows in the matrix represent strategies for player 1 and columns for player 2. In each entry of the matrix, the ith number is the payoff for player i.

random (mixed) strategy. If we denote the space of probability distributions over a set A by $\Delta(A)$, a randomization by a player over his pure strategies is denoted by $\sigma^i \in \Sigma^i \equiv \Delta(A^i)$.

2.2. Equilibrium Concepts

In decision problems with only one decision maker, it is adequate to maximize the expected utility of the decision maker. However, in games there are many players and we need to define more elaborated solution concepts. Next we will shortly present two relevant solution concepts of matrix games.

Definition 2. *If N is the number of players, the strategies $\sigma_*^1, \ldots, \sigma_*^N$ constitute a* Nash equilibrium *solution of the game if the following inequality holds for all $\sigma^i \in \Sigma^i$ and for all i:*

$$r^i(\sigma_*^1, \ldots, \sigma_*^{i-1}, \sigma^i, \sigma_*^{i+1}, \ldots, \sigma_*^N) \leq r^i(\sigma_*^1, \ldots, \sigma_*^N)$$

The concept of the Nash equilibrium was originally proposed by Nash.[31] The idea of the Nash equilibrium solution is that the strategy choice of each player is a best response to his opponents' play and therefore there is no need for deviation from this equilibrium point for any player alone. Thus, the concept of a Nash equilibrium solution provides a reasonable solution concept for a matrix game when the roles of the players are symmetric. However, there are decision problems in which one of the players has the ability to enforce his strategy onto other players. For solving these kind of optimization problems we have to use a hierarchical equilibrium solution concept, that is, the *Stackelberg equilibrium* concept.[32] In the two-player case, where one player is acting as the leader (player 1) and the other as the follower (player 2), the leader enforces his strategy on the opponent and the follower reacts rationally to this enforcement.

The basic idea is that the leader selects his strategy so that he enforces the opponent to select the response that leads to the optimal response for the leader. Algorithmically, in the case of finite bimatrix games where player 1 is the leader and player 2 is the follower, obtaining a Stackelberg solution $(a_S^1, a_S^2(a^1))$ can be seen as the following two-step algorithm:

(1) $a_S^2(a^1) = \arg\max_{a^2 \in A^2} r^2(a^1, a^2)$
(2) $a_S^1 = \arg\max_{a^1 \in A^1} r^1(a^1, a_S^2(a^1))$

In step 1, the follower's strategy is expressed as a function of the leader's strategy. In step 2, the leader maximizes his own utility by selecting the optimal strategy pair. The only requirement is that the follower's response is unique; if this is not the case, some additional restrictions must be set.

As an example of the Stackelberg equilibrium solution concept, let us consider the game in Figure 3. If we now stipulate that player 1 is the leader and player 2 is the follower, player 1 can enforce his strategy on player 2. Thus, before player 1 enforces his strategy on player 2, he has to consider the possible responses

	a_1^2	a_2^2	a_3^2
a_1^1	**0,1**	**-2,-1**	$-\frac{3}{2}, \frac{2}{3}$
a_2^1	**-1,-2**	**-1,0**	**-3,-1**
a_3^1	**1,0**	**-2,-1**	$-2, \frac{1}{2}$

Figure 3. An example matrix game. Its unique Nash equilibrium in pure strategies is (a_2^1, a_2^2). If player 1 is the leader, the Stackelberg equilibrium solution is (a_1^1, a_1^2). Correspondingly, if player 2 is the leader, (a_1^1, a_3^2) is the Stackelberg equilibrium solution.[33]

of player 2 and then enforce the strategy that is the most favorable to him. In this example game, if player 1 chooses the strategy a_1^1, then the best response of player 2 is a_1^2. If player 1 chooses a_2^1, then the follower's response is a_2^2, and if player 1 chooses a_3^1, player 2 responds using strategy a_3^2. Clearly, player 1 maximizes his own utility by choosing strategy a_1^1 and thus the Stackelberg equilibrium solution is (a_1^1, a_1^2).

3. SINGLE-AGENT REINFORCEMENT LEARNING

In this section, we briefly introduce the mathematical theory of noncompetitive Markov decision processes. In addition, practical solution methods for these processes are discussed at the end of this section.

3.1. Markov Decision Process

A fundamental concept in a *Markov Decision Process* is an *agent* that interacts with the environment in the manner illustrated in Figure 4. The environment

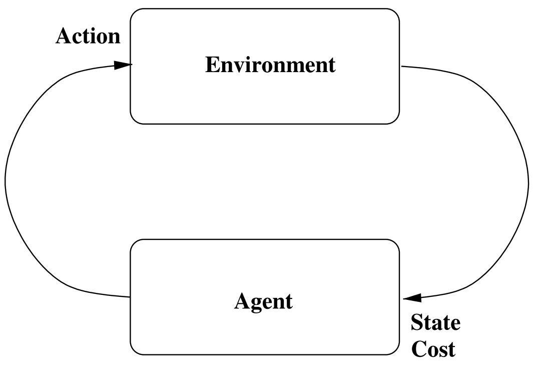

Figure 4. An overview of the learning system.

evolves (changes its state) probabilistically, and for each state there is a set of possible actions that the agent may take. Every time the agent takes an action, a certain cost is incurred. The concept of MDP was originally proposed by Bellman[34] as a discretization of the optimal control problem.

Formally, we define the Markov decision process as follows.

DEFINITION 3. *A* Markov Decision Process *(MDP) is a tuple* (S, A, p, r)*, where S is the set of all states, A is the set of all actions,* $p : S \times A \to \Delta(S)$ *is the state transition function, and* $r : S \times A \to \mathbb{R}$ *is the reward function.* $\Delta(S)$ *is the set of probability distributions over the set S.*

Additionally, we need a *policy*, that is, a rule stating what to do, given the knowledge of the current state of the environment. The policy is defined as a function from states to actions:

$$\pi : S_t \to A_t \tag{1}$$

where t refers to the discrete time step. The policy is *stationary* if there are no time dependencies, that is,

$$\pi : S \to A \tag{2}$$

In this article, we are only interested in stationary policies. The goal of the agent is to find the policy π_* that maximizes his expected discounted utility R:

$$V_\pi(s) = E_\pi[R|s_0 = s] = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right] \tag{3}$$

where r_t is an immediate reward at time step t and γ is a discount factor. Moreover, the value for each state-action pair is:

$$Q_\pi(s,a) = E_\pi[R|s_0 = s, a_0 = a] = r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_\pi(s') \tag{4}$$

Finding the optimal policy π_* can be seen as an optimization problem, which can be solved, for example, using dynamic programming algorithms.

3.2. Solving MDPs

Using dynamic programming requires solving the following equation for all states $s \in S$:

$$V_{\pi_*}(s) = \max_{a \in A(s)} Q_{\pi_*}(s,a) \tag{5}$$

These equations, *Bellman optimality equations*, form a basis for reinforcement learning algorithms. There are two basic methods for calculating the optimal policy, *policy iteration* and *value iteration* (for detailed discussion, see, e.g., Ref. 35). In the policy iteration algorithm, the current policy is evaluated and then improved using greedy optimization based on the evaluation step. The value iteration algorithm is based on successive approximations of the value function and there is no need for repeated computation of the exact value function.

In both algorithms, the exact model of the environment should be known a priori. In many situations, however, we do not have the model available. Fortunately, it is possible to approximate the model from individual samples on-line. These methods are called temporal difference methods and can be divided into off-policy and on-policy methods based on whether they are using the same policy they are optimizing for learning or not. An example of on-policy methods is SARSA, which has the update rule[36]

$$Q_{t+1}(s_t,a_t) = (1-\alpha_t)Q_t(s_t,a_t) + \alpha_t[r_{t+1} + \gamma Q_t(s_{t+1},b)] \tag{6}$$

where the action selection in the state s_{t+1} occurs according to the current policy. An example of off-policy methods is Q-learning. Its update rule is[37,38]

$$Q_{t+1}(s_t,a_t) = (1-\alpha_t)Q_t(s_t,a_t) + \alpha_t\left[r_{t+1} + \gamma \max_{b\in A} Q_t(s_{t+1},b)\right] \tag{7}$$

When an off-policy method like Q-learning is used, some policy other than the one being optimized can be used for exploring the state space. Therefore off-policy methods are more widely used in real applications.

4. MULTIAGENT REINFORCEMENT LEARNING

Until now, we have only discussed the case where there is only one agent in the environment. In this section we extend the theory of MDPs to the case of multiple decision makers in the same environment. At the end of the section, a number of solving and learning methods for this extended model are briefly discussed.

4.1. Markov Games

With multiple agents in the environment, the fundamental problem of single-agent MDPs is that the approach treats the other agents as a part of the environment and thus ignores the fact that the decisions of the other agents may influence the state of the environment.

One possible solution is to use competitive multiagent Markov decision processes, that is, *Markov games*. In a Markov game, the process changes its state according to the action choices of all the agents and can thus be seen as a multicontroller Markov decision process. Formally, we define a Markov game as follows.

DEFINITION 4. *A* Markov game *(*stochastic game*) is defined as a tuple* $(S, A^1, \ldots, A^N, p, r^1, \ldots, r^N)$, *where N is the number of agents, S is the set of all states,* A^i *is the set of all actions for each agent* $i \in [1,N]$, $p: S \times A^1 \times \cdots \times A^N \to \Delta(S)$ *is the state transition function, and* $r^i: S \times A^1 \times \cdots \times A^N \to \mathbb{R}$ *is the reward function for agent i.* $\Delta(S)$ *is the set of probability distributions over the set S.*

The concept of Markov game was originally proposed by Shapley.[39] Again, as in the case of single-agent MDP, we need a policy π^i for each agent i (the policies are assumed to be stationary):

$$\pi^i : S \to A^i, \forall i \in \{1, N\} \tag{8}$$

In multiagent systems this policy function is not necessarily deterministic. However, here we assume that the randomization is performed inside the policy function and therefore π^i returns actions directly. The expected value of the discounted utility R^i for agent i is the following:

$$V^i_{\pi^1,\ldots,\pi^N}(s) = E_{\pi^1,\ldots,\pi^N}[R^i | s_0 = s] = E_{\pi^1,\ldots,\pi^N}\left[\sum_{t=0}^{\infty} \gamma^t r^i_{t+1} | s_0 = s\right] \tag{9}$$

where r^i_t is the immediate reward at time step t for agent i and γ is a discount factor. Moreover, the value for each state-action pair is

$$\begin{aligned} Q^i_{\pi^1,\ldots,\pi^N}(s, a^1, \ldots, a^N) &= E_{\pi^1,\ldots,\pi^N}[R^i | s_0 = s, a^1_0 = a^1, \ldots, a^N_0 = a^N] \\ &= r^i(s, a^1, \ldots, a^N) \\ &\quad + \gamma \sum_{s'} p(s' | s, a^1, \ldots, a^N) V^i_{\pi^1,\ldots,\pi^N}(s') \end{aligned} \tag{10}$$

In contrast to the single-agent MDP, finding the optimal policy π^i_* for each agent i can be seen as a game theoretical problem where the strategies the players can choose are the policies defined in Equation 8.

4.2. Solving Markov Games

In the case of multiagent reinforcement learning, it is not enough to maximize the expected utilities of individual agents. Instead, our goal is to find an equilibrium policy of the Markov game, for example, a Nash equilibrium policy. The Nash equilibrium policy is defined as follows.

DEFINITION 5. *If N is the number of agents and Π^i is the policy space for agent i, the policies $\pi^1_*, \ldots, \pi^N_*$ constitute a Nash equilibrium solution of the game if the following inequality holds for all $\pi^i \in \Pi^i$ and for all i in each state $s \in S$:*

$$V^i_{\pi^1_*,\ldots,\pi^i,\ldots,\pi^N_*}(s) \leq V^i_{\pi^1_*,\ldots,\pi^N_*}(s)$$

It is noteworthy that Definition 5 coincides with Definition 2 when individual strategies are replaced with policies. The Stackelberg equilibrium concept can be extended for policies in similar fashion. We refer to methods build on Markov games with the Nash equilibrium concept as symmetric methods and to methods that utilize the Stackelberg equilibrium concept as asymmetric methods.

If the exact model, that is, rewards and state transition probabilities, is known a priori, it is possible to solve the game using standard mathematical optimization methods. However, only a few special cases of Markov games can be solved with linear programming and, in general, more complex methods are needed.

4.3. Symmetric Learning in Markov Games

As in the case of single agent reinforcement learning, Q-values defined in Equation 10 can be learned from observations on-line using some iterative algorithm. For example, in the two-agent case, if we use Q-learning, the update rule for agent 1 is[16]

$$Q^1_{t+1}(s_t, a^1_t, a^2_t) = (1 - \alpha_t) Q^1_t(s_t, a^1_t, a^2_t) + \alpha_t [r^1_{t+1} + \gamma \mathrm{Nash}\{Q^1_t(s_{t+1})\}] \tag{11}$$

where $\mathrm{Nash}\{Q^1_t(s_{t+1})\}$ is the Nash equilibrium outcome of the bimatrix game defined by the payoff functions $Q^1_t(s_{t+1})$ and $Q^2_t(s_{t+1})$. The corresponding update rule for agent 2 is[16]

$$Q^2_{t+1}(s_t, a^1_t, a^2_t) = (1 - \alpha_t) Q^2_t(s_t, a^1_t, a^2_t) + \alpha_t [r^2_{t+1} + \gamma \mathrm{Nash}\{Q^2_t(s_{t+1})\}] \tag{12}$$

Note that it is guaranteed that every finite matrix game possesses at least one Nash equilibrium in mixed strategies. However, there exists a not necessarily Nash equilibrium point in pure strategies and therefore $\mathrm{Nash}\{Q^1_t(s_{t+1})\}$ in Equation 11 returns the value of a mixed strategy equilibrium.

4.4. Asymmetric Learning in Markov Games

A Markov game can be seen as a set of matrix games associated with each state $s \in S$. If the value functions of both the leader and the follower are known, we can obtain an asymmetric solution of the Markov game by solving the matrix game associated with each state s using the Stackelberg equilibrium solution concept. The following three-stage protocol solves a Stackelberg equilibrium solution in a state $s \in S$:

(1) Determination of the cooperation strategies $a^c = (a^{1c}, a^{2c})$ by finding the maximum element of the matrix game $Q^1_{\pi_1,\pi_2}$ in the state s:

$$\arg \max_{\substack{a^1 \in A^1 \\ a^2 \in A^2}} Q^1_{\pi^1,\pi^2}(s, a^1, a^2) \tag{13}$$

(2) Determination of the leader's enforcement (and action, $a^1_S = g(s, a^c)$):

$$g(s, a^c) = \arg \min_{a^1 \in A^1} \| f(Q^2_{\pi^1,\pi^2}(s, a^1, a^2)), a^c \| \tag{14}$$

(3) Determination of the follower's response a_S^2:

$$a_S^2 = \arg\max_{a^2 \in A^2} Q^2_{\pi^1,\pi^2}(s, g(s,a^c), a^2) \tag{15}$$

In the protocol, $\|a, a^c\|, a \in A^2$ is a distance measure, defined in the Q-value space of the leader, measuring the distance between the Q-value corresponding to a particular action and the Q-value associated with the cooperation strategies (maximal possible payoff for the leader), that is,

$$\|x, a^c\| = |Q^1_{\pi^1,\pi^2}(s, a^1, x) - Q^1_{\pi^1,\pi^2}(s, a^{1c}, a^{2c})| \tag{16}$$

where a^1 is the action selected by the leader in step 2. The function f is used to select actions by player 2, for example, in the case of of greedy action selection $f = \arg\max_{a^2 \in A^2}$. In practical implementations of the protocol, for example, when the protocol is applied to action selection during learning, the minimization in step 2 can be replaced with the *softmin* function and the maximization in step 3 with the *softmax* function for ensuring the proper exploration of the state-action space; see Ref. 35.

Actual learning of the payoffs $Q^1_{\pi^1_S,\pi^2_S}$ and $Q^2_{\pi^1_S,\pi^2_S}$ can be done by using any suitable method from the field of reinforcement learning. In this article we present the equations for asymmetric multiagent Q-learning. If agent 1 is the leader and agent 2 is the follower, update rules for the Q-values are as follows:

$$\begin{aligned} Q^1_{t+1}(s_t, a^1_t, a^2_t) &= (1-\alpha_t) Q^1_t(s_t, a^1_t, a^2_t) \\ &\quad + \alpha_t [r^1_{t+1} + \gamma \max_{b \in A^1} Q^1_t(s_{t+1}, b, Tb)] \end{aligned} \tag{17}$$

$$\begin{aligned} Q^2_{t+1}(s_t, a^1_t, a^2_t) &= (1-\alpha_t) Q^2_t(s_t, a^1_t, a^2_t) \\ &\quad + \alpha_t [r^2_{t+1} + \gamma \max_{b \in A^2} Q^2_t(s_{t+1}, g(s_{t+1}, a^c_{t+1}), b)] \end{aligned} \tag{18}$$

where $g(s_t, a^c_t)$ is the leader's enforcement and T is a mapping $T: A^1 \to A^2$ that conducts the follower's best response to the leader's enforcement. Note that both agents are able to build a model of their opponent if the state, action, and reward information is fully observable to the agents. For example, the leader can build a model of the follower by estimating Q^2 function during the learning process. Moreover, if the leader always selects the same equilibrium strategy in each state, the value of each stage game is unique. More thorough treatment of asymmetric multiagent reinforcement learning can be found in Ref. 26.

5. NUMERICAL SOLUTION METHODS FOR MULTIAGENT REINFORCEMENT LEARNING

In this section we propose two numerical solution methods for the multiagent learning models discussed above. The first method approximates the Q-function with an arbitrary function approximator and the second method uses a parameterized policy function family. The latter model is suitable especially for the asymmetric learning model in team Markov games.

5.1. Value-Function-Based Gradient Method

In the first method, the value-function-based gradient method, we utilize a parameterized approximation of the Q-function. In single-agent domains, the corresponding method was originally proposed by Baird.[40] Let us denote this approximation for agent i as $Q^i(s, a^1, a^2; \boldsymbol{\omega}^i)$, where $\boldsymbol{\omega}^i$ is a vector containing parameters for the approximating function. At the time instance t, the error, the *squared Bellman residual*, of agent i is as follows:

$$e_t^i = \frac{1}{2}[r_{t+1}^i + \gamma W(s_{t+1}; \boldsymbol{\omega}^1, \boldsymbol{\omega}^2) - Q_t^i(s_t, a_t^1, a_t^2; \boldsymbol{\omega}^i)]^2 \tag{19}$$

where $W(s_t)$ is an operator returning the value of the state s_t, for example, the value of a Nash or Stackelberg equilibrium point of the state s_t.

The modified gradient of the error function with respect to an arbitrary parameter ω in $\boldsymbol{\omega}^i$ is the following:

$$\begin{aligned} \frac{\partial e_t^i}{\partial \omega} &= [r_{t+1}^i + \gamma W(s_{t+1}; \boldsymbol{\omega}^1, \boldsymbol{\omega}^2) - Q_t^i(s_t, a_t^1, a_t^2; \boldsymbol{\omega}^i)] \\ &\quad \times \left[\xi\gamma \frac{\partial W(s_{t+1}; \boldsymbol{\omega}^1, \boldsymbol{\omega}^2)}{\partial \omega} - \frac{\partial Q_t^i(s_t, a_t^1, a_t^2; \boldsymbol{\omega}^i)}{\partial \omega}\right] \end{aligned} \tag{20}$$

where $\xi \in [0,1]$ is a parameter controlling the direction of the modified gradient. If we set $\xi = 1$, we have the real gradient of Equation 19. However, the learning could be extremely slow in this case and thus it is favorable to use values of less than one for ξ. With $\xi < 1$, Equation 20 is not the real gradient but the learning is usually much faster than with the gradient. Note that in the extreme case $\xi = 0$, Equation 20 reduces to

$$\frac{\partial e_t^i}{\partial \omega} = -[r_{t+1}^i + \gamma W(s_{t+1}; \boldsymbol{\omega}^1, \boldsymbol{\omega}^2) - Q_t^i(s_t, a_t^1, a_t^2; \boldsymbol{\omega}^i)] \frac{\partial Q_t^i(s_t, a_t^1, a_t^2; \boldsymbol{\omega}^i)}{\partial \omega} \tag{21}$$

which is, in fact, a form of the *Widrow–Hoff* learning rule.

It is possible to derive theoretical limits for ξ that guarantee the convergence of the modified gradient method with arbitrary function approximators (see, e.g., Ref. 40), but selecting ξ heuristically is often enough. For example, it is favorable that the system learns very fast in the beginning of the learning process and therefore it is justified to set ξ initially as small as possible and then gradually increase the value of ξ during the learning process.

5.2. Direct Policy Gradient Methods

In the previous method, the Q-function was approximated with a parameterized function approximator. Another approach is to directly use the function approximator in the policy function. In this section we present a method suitable for team Markov games, that is, systems where all the agents share the same utility function. The corresponding single-agent method was proposed by Sutton et al.[11]

The main idea of the method is to approximate the policy function with some parameterized function approximator. Formally, we denote the parameterized *joint policy function* as

$$\pi(s, a^1, a^2; \boldsymbol{\theta}) = P(a^1, a^2 | s; \boldsymbol{\theta}) \tag{22}$$

where $\boldsymbol{\theta}$ is an arbitrary parameter vector. In other words, the joint policy function defines a probability distribution over the joint action space of the agents in each state. The value of the state with respect to this distribution is:

$$V(s) = \sum_{a^1 \in A^1} \sum_{a^2 \in A^2} \pi(s, a^1, a^2; \boldsymbol{\theta}) Q_\pi(s, a^1, a^2) \tag{23}$$

Now our goal is to find a joint policy function that maximizes the discounted value $V(s)$ when we start from an arbitrary state s. We denote our objective function as $\rho(s, \pi)$ and differentiate it with respect to $\theta \in \boldsymbol{\theta}$, thus obtaining the gradient in the following form:

$$\frac{\partial \rho(s, \pi)}{\partial \theta} = \sum_{x \in S} d_\pi(x) \sum_{a^1 \in A^1} \sum_{a^2 \in A^2} \frac{\partial \pi(x, a^1, a^2; \boldsymbol{\theta})}{\partial \theta} Q_\pi(x, a^1, a^2) \tag{24}$$

where $d_\pi(x)$ is the discounted probability of reaching state x when the joint policy π is followed infinitely long. Normally we do not know the exact model of the system, and therefore it is not possible to calculate this probability value. However, if the states s are sampled from the policy π, we get an unbiased estimate of Equation 24 even if we neglect the probability term $d_\pi(x)$ from this equation.

Furthermore, the Q-function is not normally known a priori. We approximate this function with an approximator $f(s, a^1, a^2; \boldsymbol{\omega})$, where $\boldsymbol{\omega}$ is an arbitrary parameter vector. We also require that f satisfies the following compatibility property:

$$\frac{\partial f(s, a^1, a^2, \boldsymbol{\omega})}{\partial \omega} = \frac{\partial}{\partial \theta} \ln \pi(s, a^1, a^2, \boldsymbol{\theta}) \tag{25}$$

The purpose of the above equation is to require that the error in f is orthogonal to the gradient of the policy function. For example, in this article we use a version of the Gibbs distribution as a policy function, that is,

$$\pi(s, a^1, a^2; \boldsymbol{\theta}) = \frac{e^{\boldsymbol{\theta}^T \phi(s, a^1, a^2)}}{\sum_{b \in A^1} \sum_{c \in A^2} e^{\boldsymbol{\theta}^T \phi(s, b, c)}} \tag{26}$$

where $\phi(s, a^1, a^2)$ is a unit vector with the parameter corresponding to tuple (s, a^1, a^2) set to one. Hence, the compatible function approximator is

$$f(s, a^1, a^2, \boldsymbol{\omega}) = \boldsymbol{\omega}^T \left[\phi(s, a^1, a^2) - \sum_{b \in A^1} \sum_{c \in A^2} \phi(s, b, c) \pi(s, b, c; \boldsymbol{\theta}) \right] \tag{27}$$

Note that Equation 27 represents the advantage of choosing a pair (a^1, a^2) in the state s, and therefore it is possible to teach the weights $\boldsymbol{\omega}$ by using some standard technique such as Q-learning.

5.3. Convergence Properties

If the parameter $\xi = 1$, Equation 20 is the true gradient of Equation 19. However, convergence proofs can only be provided if the operator W brings Q-function closer to an equilibrium Q-values in each iteration of the algorithm; that is, the operator W is a *pseudo-contraction*. Example operators for which the contraction property holds are *Max* leading to the single-agent Q-learning, *MaxMin* (see Ref. 15) for zero-sum Markov games and *MaxMax* (see Ref. 29) for team games. Operators calculating Nash or Stackelberg outcome of a matrix game are not pseudo-contractions in general-sum problems but can still lead to good performance (see, e.g., Refs. 26 and 41).

With pseudo-contraction operators, the convergence of Q-learning type temporal-difference methods is guaranteed from arbitrary initial values if all state-action tuples are visited infinitely many times. The actual learning speed is heavily dependent on the initial Q-values. Usually right mutual ordering of the actions is found much earlier than the exact Q-values.

6. A PRICING PROBLEM

In this section, we apply the above discussed numerical learning methods to a pricing problem. In the problem, there are two competing agents (brokers) that sell identical products and compete against each other on the basis of price. At each time step, one of the brokers decides its new price based on the opponent's, that is, other broker's, current price. After the prices have been set, the customer either buys a product from the seller or decides not to buy the product at all. The objective of the agents is to maximize their profits. We begin the section by modeling the interaction between the two brokers as an asymmetric multiagent reinforcement learning problem.

6.1. Problem Overview

In Ref. 1, Tesauro and Kephart modeled the interaction between two brokers as a single-agent reinforcement learning problem in which the goal of the learning agent is to find the pricing strategy that maximizes its long-time profits. Additionally, reinforcement learning aids the agents to prevent "price wars," that is, repeated price undercutting among the brokers. As a consequence of a price war, the prices would become very low and the overall profits would also be small. Tesauro and Kephart reported very good performance of the approach when one of the brokers keeps its pricing strategy fixed. However, if both brokers try to learn simultaneously, the Markov property assumed in the theory of MDPs does not hold any more and the learning system encounters serious convergence problems. In this article, we model the system as a Markov game and solve it by two learning methods. Further, we test the proposed learning system with two economic models.

In the simpler economic model (the Shopbot model[42]), the customer buys the product from the broker with the lowest price. At each time step, after the customer has made his purchase decision, both brokers get their immediate profits according to the utility functions defined as follows:

$$u^1(p^1,p^2) = \begin{cases} p^1 - c & \text{if } p^1 \le p^2 \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

and

$$u^2(p^1,p^2) = \begin{cases} p^2 - c & \text{if } p^1 > p^2 \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

where $p^1, p^2 \in P$ are the current prices of broker 1 and broker 2, respectively, and the $c \in [0,1]$ is a fixed marginal cost of the product. In this article, all prices lie in the unit interval and the parameter $c = 0.2$.

In the second, more complex economic model (the Price–Quality model[43]), there is a quality parameter associated with each broker, and the customers make their purchase decisions in a quality-aware manner. Denoting the quality parameter for broker 1 as q^1 and for broker 2 as q^2, we get the following utility functions for the brokers[43]:

$$u^1(p^1,p^2) = \begin{cases} (q^1 - p^1)(p^1 - c(q^1)) & \text{if } 0 \le p^1 \le p^2 \\ & \text{or } p^1 > q^2 \\ (q^1 - q^2)(p^1 - c(q^1)) & \text{if } p^2 < p^1 < q^2 \end{cases} \tag{30}$$

and

$$u^2(p^1,p^2) = \begin{cases} (q^2 - p^2)(p^2 - c(q^2)) & \text{if } 0 \le p^2 < p^1 \\ 0 & p^2 \ge p^1 \end{cases} \tag{31}$$

where $c(q^i)$ represents the cost of producing the product i. Note that we assume here that there are an infinite number of customers who all have a minimum quality threshold and a maximum price threshold. Hence, the above utility functions are simply profit expectations for the brokers. In this work, we use the following linear cost function:

$$c(q^i) = 0.1(1.0 + q^i) \tag{32}$$

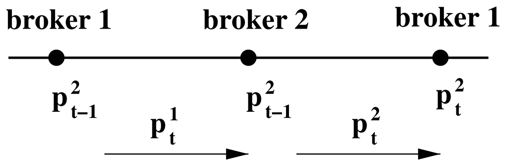

Figure 5. Timeline of the price decisions in the pricing problem. Price symbols below dots describe states and symbols above arrows price decisions.

Furthermore, we set the quality parameters as follows: $q^1 = 1.0$ and $q^2 = 0.9$. This parameter setting was observed to generate price wars when the agents use the simple myopic pricing strategy (i.e., they make their decisions directly based on the above declared utility functions) in Ref. 1.

6.2. Associated Markov Game

We make the assumption that the brokers do not decide their decisions simultaneously; that is, there is an ordering among the decision makers. Hence, we model the system with the following Markov game endowed with the asymmetric equilibrium concept:

- The state is the current price of broker 2.
- Broker 1 is acting as the leader and hence decides its price prior to broker 2. Hence, as the state is the current price of broker 2, the utility of broker 1 depends only on its price selection and the current state.
- Broker 2 is the follower and its utility value depends on the leader's enforcement and its own price selection.

At each time step, broker 1 calculates a Stackelberg equilibrium point of the matrix game associated with the current state and makes its pricing decision based on this solution. After that, broker 1 announces its price decision to broker 2 who, in its turn, maximizes its utility value based on this enforcement. This process is illustrated in Figure 5. The corresponding update equations for brokers 1 and 2 are as follows:

$$\begin{aligned} Q^1_{t+1}(p^2_{t-1}, p^1_t, p^2_t) &= (1-\alpha_t) Q^1_t(p^2_{t-1}, p^1_t, p^2_t) \\ &\quad + \alpha_t \left[u^1(p^1_t, p^2_{t-1}) + \gamma \max_{b \in P} Q^1_t(p^2_t, b, Tb) \right] \end{aligned} \tag{33}$$

and

$$\begin{aligned} Q^2_{t+1}(p^2_{t-1}, p^1_t, p^2_t) &= (1-\alpha_t) Q^2_t(p^2_{t-1}, p^1_t, p^2_t) \\ &\quad + \alpha_t \left[u^2(p^1_t, p^2_t) + \gamma \max_{b \in P} Q^2_t(p^2_t, g(p^2_t, a^c_t), b) \right] \end{aligned} \tag{34}$$

where γ is the discount factor, operator *Tb* conducts the follower's response to the leader's action choice *b*, and $g(\cdot)$ is the leader's enforcement. Note that the learning method does not need any prior model of the opponent, not even the above defined utility functions.

6.3. Teaching the Model with the Value-Function-Based Gradient Method

At first, we teach the model by using directly the update rules in Equations 33 and 34. Note that this is the same as Equation 20, if we use a linear function approximator of the form

$$Q^i(s, a^1, a^2) = (\boldsymbol{\omega}^i)^T \phi(s, a^1, a^2) \tag{35}$$

for Q-function approximation and set the parameter $\xi = 0$. In Equation 35, $\phi(s, a^1, a^2)$ is a unit vector with the parameter corresponding to the tuple (s, a^1, a^2) set to one.

In our test runs, the number of different pricing options was 15 for both agents. During training each state-actions tuple was visited 1000 times. Learning rate parameter α was decayed linearly during the learning.

In the testing phase, the initial state (price of broker 2) was selected randomly and one test run consisted of 10 pricing decisions per broker. In Figure 6, the cumulative profits (average from 1000 test runs) of both agents are plotted against

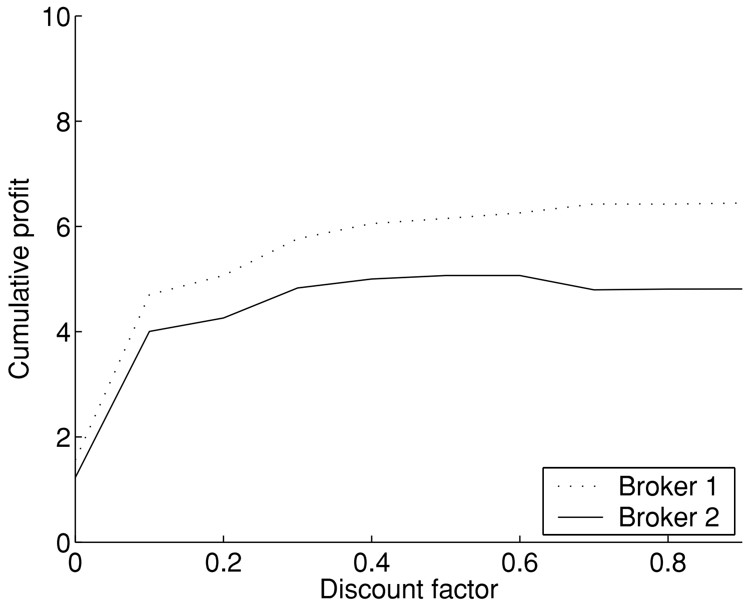

Figure 6. Averaged profits with the Shopbot economic model. All data points are averages of 1000 test runs, each containing 10 pricing decisions for both agents.

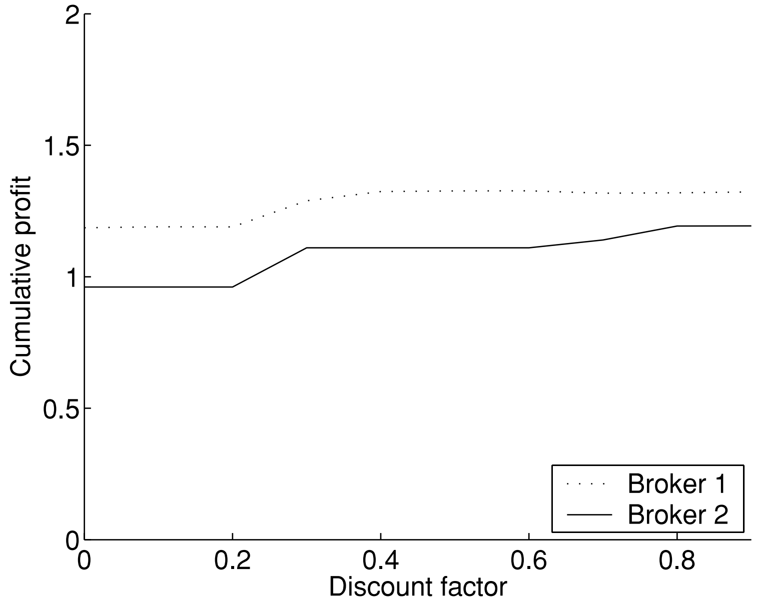

Figure 7. Averaged profits with the Price-Quality economic model. All data points are averages of 1000 test runs, each containing 10 pricing decisions for both agents.

the discount factor γ in the case of the Shopbot pricing model. Respectively, in Figure 7 the cumulative profits are shown in the case of the Price-Quality model. With the Shopbot model, the average profit of broker 1 grows monotonically as the discount factor increases. Also the profit of broker 2 increases albeit not monotonically. Moreover, the use of small discount factor $\gamma = 0.1$, corresponding to very shallow lookahead, leads to relatively high profits compared to $\gamma = 0.0$. The use of higher discount factors increases profits further but the growth is not so dramatic. With the Price-Quality model, the profits grow steadily as the discount factor is increased. However, the growth is not very large, which implies that it is not necessary to use a large discount factor with this pricing model.

The convergence of the agents' Q-values ($\boldsymbol{\omega}^i$) in the case of Shopbot model is illustrated in Figure 8, where the Euclidean distance between vectors containing values from consecutive training rounds is plotted against the round number. Two cases with discount factors 0.3 and 0.9 are plotted for both brokers. It can be seen that the algorithm converged very fast in every case. With high γ values the convergence is much slower than with low values of γ. The convergence properties of the algorithm in the case of the Price-Quality model are analogous, as can be seen in Figure 9.

6.4. Teaching the Model with the Policy Gradient Method

Albeit the pricing model discussed in this article is not a team game, that is, interests of the brokers are partially conflicting, there is a very strong correlation

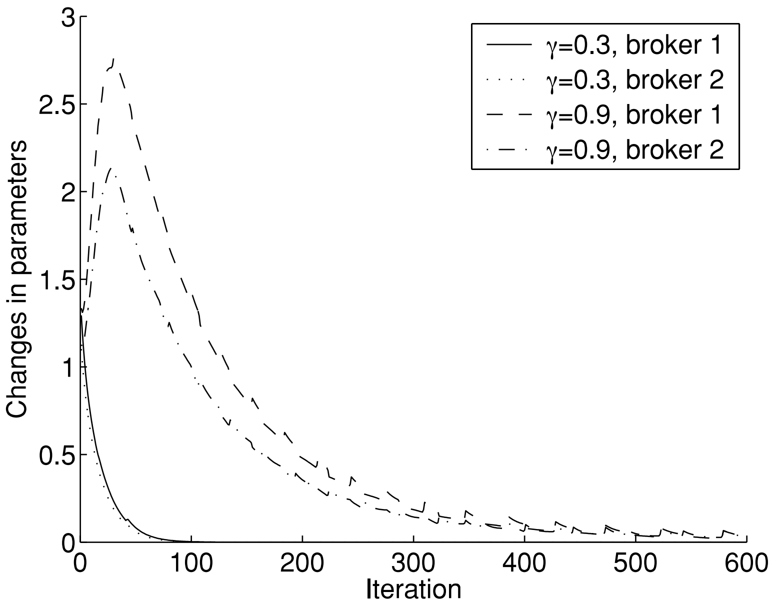

Figure 8. Convergence of the Q-values with the Shopbot economic model.

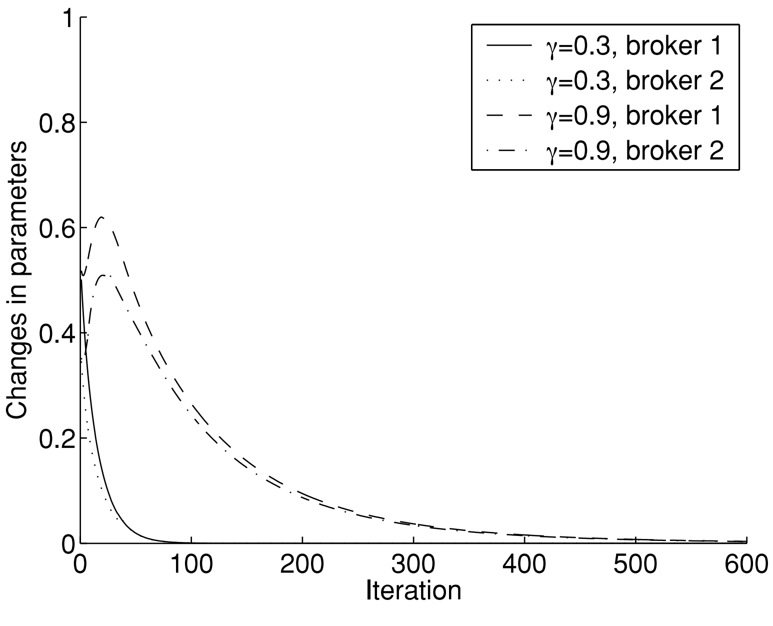

Figure 9. Convergence of the Q-values with the Price-Quality economic model.

between utilities of the brokers. This correlation is mainly due to the fact that if the broker sets a very high price, it is also possible for its competitor to set a high price.

We apply the policy gradient method to the pricing problem by allowing both agents to update the probability distribution π by using Equation 24. Formally, agent i uses the policy gradient update rule with its Q-function:

$$\frac{\partial \rho^i(s,\pi)}{\partial \theta} = \sum_{x \in S} d_\pi(x) \sum_{a^1 \in A^1} \sum_{a^2 \in A^2} \frac{\partial \pi(x, a^1, a^2; \boldsymbol{\theta})}{\partial \theta} Q^i_\pi(x, a^1, a^2) \tag{36}$$

Now, as both agents update the same joint distribution and the learning rate parameter α is the same for both agents, the weights are changed according to

$$\Delta\theta \propto \sum_{a^1 \in A^1} \sum_{a^2 \in A^2} \frac{\partial \pi(s_t, a^1, a^2; \boldsymbol{\theta})}{\partial \theta} [Q^1_\pi(s_t, a^1, a^2) + Q^2_\pi(s_t, a^1, a^2)] \tag{37}$$

which shows that our goal is to find a policy that maximizes the summed utility of the brokers.

The state-actions space was explored by using the Gibbs-distribution defined in Equation 26. The training consisted of 10,000 episodes, each containing 10 pricing decisions. Moreover, the training procedure was repeated 20 times, and all presented results are averages from these training rounds. The testing phase consisted of 20 test runs (1 per each training round) each containing 1000 episodes of length 10 pricing decisions for both brokers. Initial prices of broker 2 were set randomly.

In Figure 10, the profits in the case of Shopbot pricing model are presented. From this figure, it can be seen that with shallow lookahead, $\gamma \leq 0.6$, broker 1 achieves slightly lower profits than broker 2 and, overall, the profits are the same as in the myopic case. When the brokers are patient enough ($\gamma > 0.6$), the profits of broker 1 start to rise. However, the sum of the profits holds fixed, which is expected, because the goal was to find a policy that maximizes the sum of the expected profits of the brokers.

Convergence of the parameters $\boldsymbol{\theta}$ is illustrated in Figure 11. When the brokers learn only the myopic utility function, the changes take place uniformly during the learning process. With higher γ values, the large changes in $\boldsymbol{\theta}$ take place after 4000 episodes because the learning of Q-values is much slower. In Figure 12, convergence of the Q-values ($\boldsymbol{\omega}^1$) of broker 1 is illustrated. With small values of γ, the changes decay linearly with time due to the decay of the learning rate parameter. With $\gamma = 0.7$, the learning is much slower. Convergence properties of $\boldsymbol{\omega}^2$ are similar.

In Figure 13, the profits in the case of the Price-Quality economic model are presented. In that case, overall profits do not grow with the discount factor γ. However, the overall profit level is quite similar with the value-function-based methods (see Figure 7).

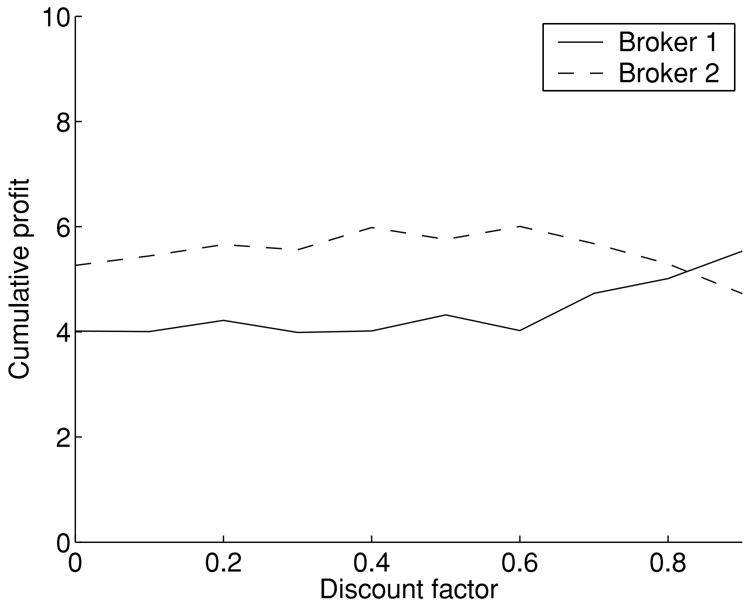

Figure 10. Averaged profits with the Shopbot economic model. All data points are averages from 20,000 test runs, each containing 10 pricing decisions for both brokers.

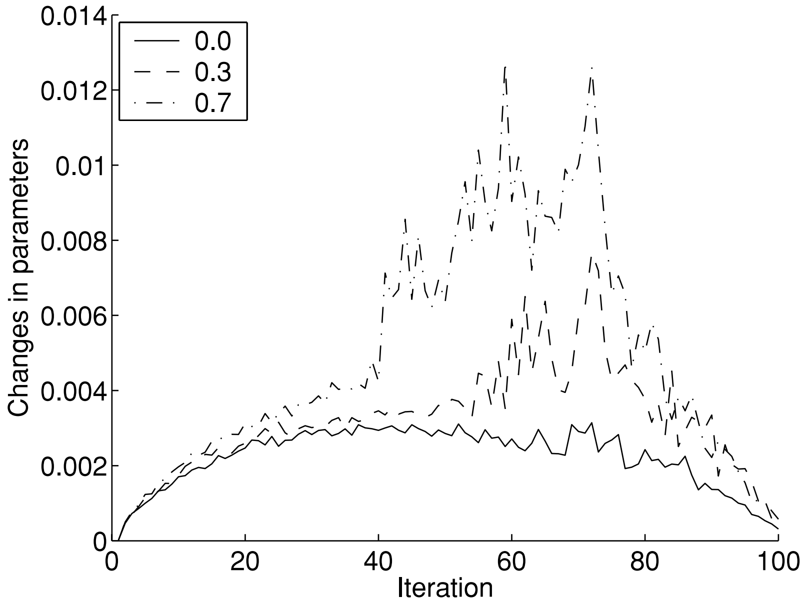

Figure 11. Convergence of the parameters $\boldsymbol{\theta}$. Values are averages from 20 training rounds. Only the value corresponding to every hundredth episode is plotted.

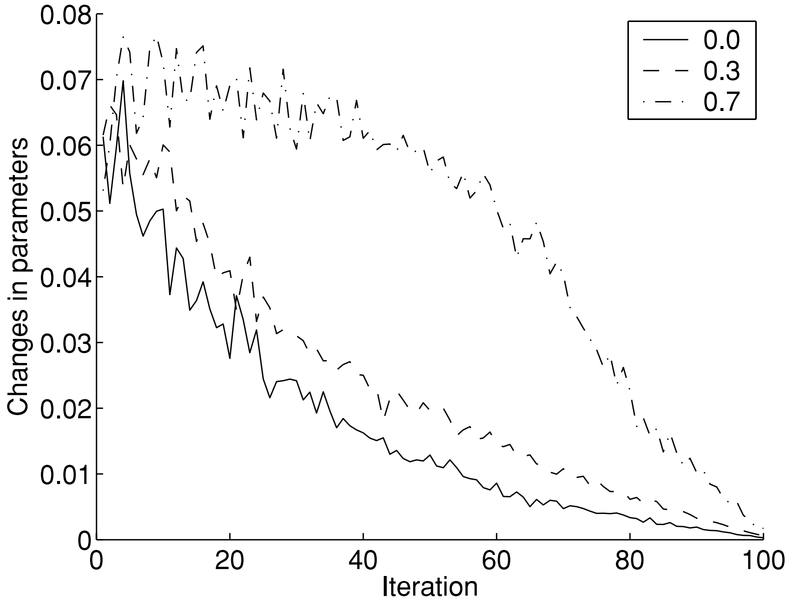

Figure 12. Convergence of the parameters $\boldsymbol{\omega}^1$. Values are averages from 20 training rounds. Only the value corresponding to every hundredth episode is plotted.

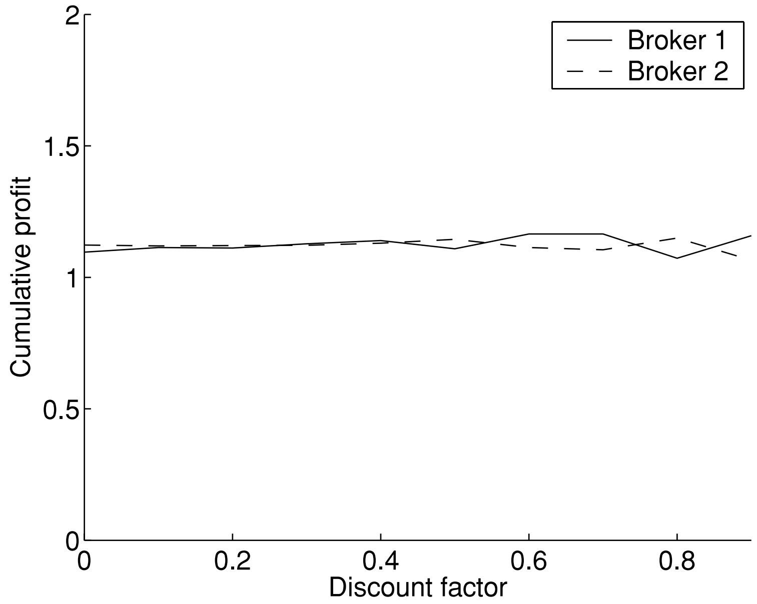

Figure 13. Averaged profits with the Price-Quality economic model. All data points are averages from 20,000 test runs, each containing 10 pricing decisions for both brokers.

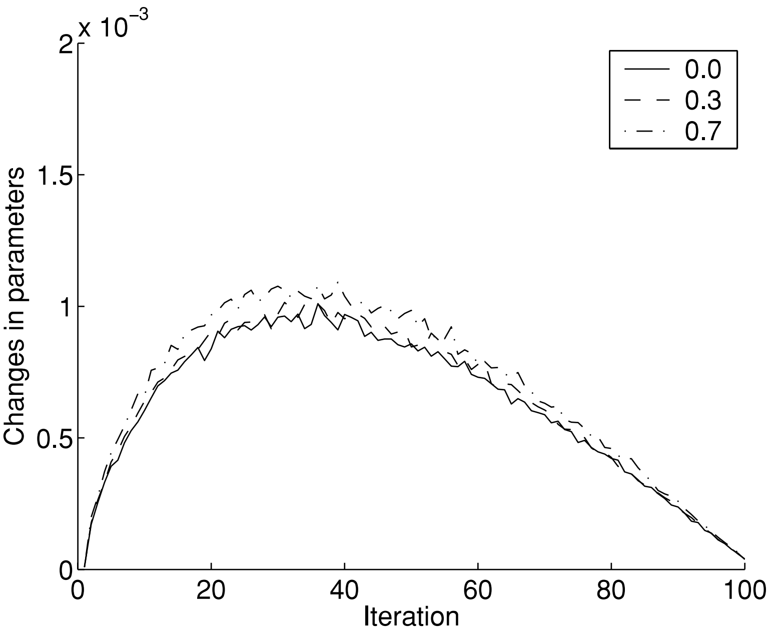

Figure 14. Convergence of the parameters $\boldsymbol{\theta}$. Values are averages from 20 training rounds. Only the value corresponding to every hundredth episode is plotted.

Convergence of the parameters $\boldsymbol{\theta}$ is illustrated in Figure 14. The curves in this figure are very smooth, indicating that the system learns only the myopic economic model (compare to the case $\gamma = 0.0$ in Figure 11). Changes in parameters $\boldsymbol{\omega}^1$ are illustrated in Figure 15. As can be seen from this figure, all curves are almost linear. The system learns slightly faster with smaller values of γ. Convergence properties of $\boldsymbol{\omega}^2$ are similar.

7. CONCLUSIONS AND FUTURE RESEARCH

A pricing problem of two competing brokers selling identical products was solved in this article by using reinforcement learning methods based on Markov games. The proposed learning methods have stronger convergence properties than single-agent reinforcement learning methods in multiagent environments. The methods converged in each test case and led to very promising results. Albeit the proposed learning methods were inherently numerical, there was still one parameter for each state-action tuple. This kind of system becomes intractable when the number of pricing options increases. Therefore an interesting extension to our current work is to apply function approximators with generalization ability, such as multilayer perceptrons, to these pricing problems.

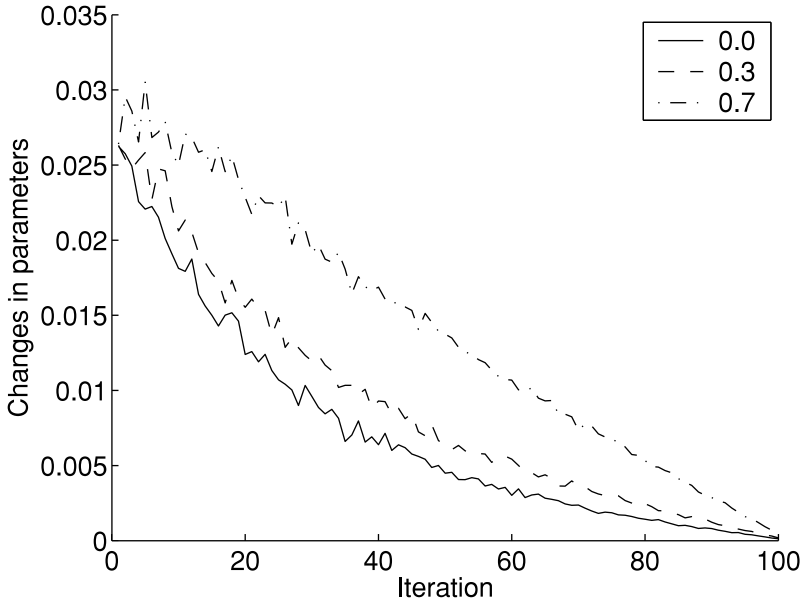

Figure 15. Convergence of the parameters $\boldsymbol{\omega}^1$. Values are averages from 20 training rounds. Only the value corresponding to every hundredth episode is plotted.

References

1. Tesauro G, Kephart JO. Pricing in agent economies using multi-agent Q-learning. In: Proc Workshop on Game Theoretic and Decision Theoretic Agents (GTDT'99), London, UK; 1999. pp 71–86.
2. Tesauro G, Kephart JO. Foresight-based pricing algorithms in an economy of software agents. In: Proc First Int Conf on Information and Computational Economics (ICE'98), Charleston, SC; 1998. pp 37–44.
3. Tesauro G. Pricing in agent economies using neural networks and multi-agent Q-learning. In: Sun R, Giles CL, editors. Sequence learning: Paradigms, algorithms, and applications, Lecture Notes in Computer Science, vol. 1828. Berlin: Springer-Verlag; 2001. pp 288–307.
4. Sridharan M, Tesauro G. Multi-agent Q-learning and regression trees for automated pricing decisions. In: Proc 17th Int Conf on Machine Learning (ICML-2000), Stanford, CA; 2000. pp 927–934.
5. Claus C, Boutilier C. The dynamics of reinforcement learning in cooperative multiagent systems. In: Proc 15th Natl Conf of Artificial Intelligence (AAAI-98), Madison, WI; 1998. pp 746–752.
6. Singh SP, Kearns M, Mansour Y. Nash convergence of gradient dynamics in general-sum games. In: Proc 16th Conf on Uncertainty in Artificial Intelligence (UAI-2000), Stanford, CA; 2000. pp 541–548.
7. Kapetanakis S, Kudenko D. Reinforcement learning of coordination in cooperative multi-agent systems. In: Proc 18th Natl Conf on Artificial Intelligence (AAAI-02), Edmonton, Alberta, Canada; 2002. pp 326–331.
8. Kapetanakis S, Kudenko D. Improving on the reinforcement learning of coordination in cooperative multi-agent systems. In: Proc AISB-2002 Symp on Adaptive Agents and Multi-Agent Systems, London, UK; 2002. pp 89–94.
9. Kapetanakis S, Kudenko D, Strens M. Learning to coordinate using commitment sequences

in cooperative multi-agent systems. In: Kudenko D, Kazakov D, Alonso E, editors. Adaptive agents and multi-agent systems II: Adaptation and multi-agent learning. Lecture Notes in Computer Science, vol. 3394. Berlin: Springer-Verlag; 2005. pp 906–918.

10. Bowling M, Veloso MM. Scalable learning in stochastic games. In: Proc AAAI-02 Workshop on Game Theoretic and Decision Theoretic Agents, Edmonton, Canada; 2002. pp 11–18.
11. Sutton RS, McAllester D, Singh SP, Mansour Y. Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems (NIPS-1999), Denver, CO; 2000. pp 1057–1063.
12. Bowling M, Veloso MM. Multiagent learning using a variable learning rate. Artif Intell 2002;136:454–460.
13. Peshkin L, Kim KE, Meuleau N, Kaelbling LP. Learning to cooperate via policy-search. In: Proc 16th Conf on Uncertainty in Artificial Intelligence (UAI-2000), Stanford, CA; 2000. pp 489–496.
14. Baird LC, Moore A. Gradient descent for general reinforcement learning. In: Advances in Neural Information Processing Systems (NIPS-1998), Denver, CO; 1999. pp 968–974.
15. Littman ML. Markov games as a framework for multi-agent reinforcement learning. In: Proc 11th Int Conf on Machine Learning (ICML-1994), New Brunswick, NJ; 1994. pp 157–163.
16. Hu J, Wellman MP. Multiagent reinforcement learning: Theoretical framework and an algorithm. In: Proc 15th Int Conf on Machine Learning (ICML-1998), Madison, WI; 1998. pp 242–250.
17. Littman ML. Friend-or-foe Q-learning in general-sum games. In: Proc 18th Int Conf on Machine Learning (ICML-2001), Williamstown, MA; 2001. pp 322–328.
18. Wang X, Sandholm TW. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In: Advances in Neural Information Processing Systems (NIPS-2002), Vancouver, British Columbia, Canada; 2003. pp 1603–1610.
19. Conitzer V, Sandholm TW. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In: Proc 20th Int Conf on Machine Learning (ICML-2003), Washington, DC; 2003. pp 83–90.
20. Greenwald A, Hall K. Correlated-Q learning. In: Proc 18th Int Conf on Machine Learning (ICML-2003), Washington, DC; 2003. pp 242–249.
21. Conitzer V, Sandholm TW. Complexity results about Nash equilibria. In: Proc Int Joint Conf on Artificial Intelligence (IJCAI-2003), Acapulco, Mexico; 2003. pp 765–771.
22. Wolpert DH, Tumer K. An introduction to collective intelligence. Technical Report NASA-ARC-IC-99-63, NASA Ames Research Center; 2000.
23. Wolpert DH, Tumer K, Frank J. Using collective intelligence to route internet traffic. In: Advances in Neural Information Processing Systems (NIPS-1998), Denver, CO; 1999. pp 952–958.
24. Wolpert DH, Kirshner S, Merz CJ, Tumer K. Adaptivity in agent-based routing for data networks. In: Proc Fourth Int Conf on Autonomous Agents (Agents 2000), Barcelona, Catalonia, Spain; 2000. pp 396–403.
25. Könönen VJ. Asymmetric multiagent reinforcement learning. In: Proc 2003 WIC Int Conf on Intelligent Agent Technology (IAT-2003), Halifax, Canada; 2003. pp 336–342.
26. Könönen VJ. Asymmetric multiagent reinforcement learning. Web Intell Agent Syst 2004;2:105–121.
27. Könönen VJ. Gradient based method for symmetric and asymmetric multiagent reinforcement learning. In: Proc Fourth Int Conf on Intelligent Data Engineering and Automated Learning (IDEAL-2003), Hong Kong, China; 2003. pp 68–75.
28. Könönen VJ. Policy gradient method for multiagent reinforcement learning. In: Proc Second Int Conf on Computational Intelligence, Robotics and Autonomous Systems (CIRAS-2003), Singapore; 2003. CD-ROM.
29. Könönen VJ. Policy gradient method for team Markov games. In: Proc Fifth Int Conf on Intelligent Data Engineering and Automated Learning (IDEAL-2004), Exeter, UK; 2004. pp 733–739.

30. Myerson RB. Game theory: Analysis of conflict. Cambridge, MA: Harvard University Press; 1991.
31. Nash JF Jr. Equilibrium points in N-person games. Proc Natl Acad Sci USA 1950;36:48–49.
32. von Stackelberg H. The theory of market economy. Oxford, UK: Oxford University Press; 1952.
33. Basar T, Olsder GJ. Dynamic noncooperative game theory. Mathematics in Science and Engineering, vol 160. London: Academic Press Inc. Ltd.; 1982.
34. Bellman RE. Dynamic programming. Princeton, NJ: Princeton University Press; 1957.
35. Sutton RS, Barto AG. Reinforcement learning: An introduction. Cambridge, MA: MIT Press; 1998.
36. Rummery GA, Niranjan M. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR166, Cambridge University, Engineering Department; 1994.
37. Watkins CJCH. Learning from delayed rewards. PhD thesis, Cambridge University; 1989.
38. Watkins CJCH, Dayan P. Q-learning. Mach Learn 1992;8:279–292.
39. Shapley LS. Stochastic games. Proc Natl Acad Sci USA 1953;39:1095–1100.
40. Baird LC. Residual algorithms: Reinforcement learning with function approximation. In: Proc 12th Int Conf on Machine learning (ICML-1995), Tahoe City, CA; 1995. pp 30–37.
41. Hu J, Wellman MP. Nash Q-learning for general-sum stochastic games. J Mach Learn Res 2003;4:1039–1069.
42. Greenwald AR, Kephart JO. Shopbots and pricebots. In: Proc IJCAI-1999 Workshop on Agent Mediated Electronic Commerce, Stockholm, Sweden; 1999. pp 1–23.
43. Sairamesh J, Kephart JO. Price dynamics of vertically differentiated information markets. In: Proc First Int Conf on Information and Computational Economics (ICE'98), Charleston, SC; 1998. pp 28–36.