Web Intelligence and Agent Systems: An international journal 3 (2005) 17–30
IOS Press

Gradient descent for symmetric and asymmetric multiagent reinforcement learning

Ville Könönen
Neural Networks Research Centre, Helsinki University of Technology, P.O. Box 5400, FI-02015 HUT, Finland
Tel.: +358 9 451 5024; Fax: +358 9 451 3277; E-mail: ville.kononen@hut.fi

Abstract. A gradient-based method for both symmetric and asymmetric multiagent reinforcement learning is introduced in this paper. Symmetric multiagent reinforcement learning addresses the problem with agents involved in the learning task having equal information states. Respectively, in asymmetric multiagent reinforcement learning, the information states are not equal, i.e. some agents (leaders) try to encourage agents with less information (followers) to select actions that lead to improved overall utility values for the leaders. In both cases, there are a huge number of parameters to learn and we thus need to use some parametric function approximation methods to represent the value functions of the agents. The method proposed in this paper is based on the VAPS framework that is extended to utilize the theory of Markov games, which is a natural basis of multiagent reinforcement learning.

Keywords: Multiagent reinforcement learning, Markov games, Nash equilibrium, Stackelberg equilibrium, value function approximation

1. Introduction

Most of the earlier work in the field of multiagent reinforcement learning deals with tabular methods and thus the proposed methods are only applicable with very limited problem instances. In this study, we propose a value function based numerical solution method for both symmetric and asymmetric multiagent reinforcement learning, which takes explicitly into account the possible changing exploration policy. The method is based on the VAPS (Value And Policy Search) framework originally proposed by Baird and Moore [1] in single-agent domains. In this paper, we extend VAPS to multiagent domains. Symmetric multiagent reinforcement learning deals with the problem where the roles of the learning systems (agents) are symmetric, i.e. the payoff function of an individual agent depends on the other agents' action choices but all agents are capable of making their decisions on their own. Asymmetric multiagent reinforcement learning addresses, in its turn, the problem where the information states of the agents involved with the learning task are not equal; some agents (leaders) try to encourage agents with less information (followers) to select actions that lead to improved payoff for the leader. This kind of configuration arises e.g. in semi-centralized multiagent systems with an external global utility associated to the system. Additionally, space and computational requirements of asymmetric learning model are often lower than the requirements in the symmetric case.

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 e.g. [6] and [23]. The method presented in [6] 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 [12] and [11]. Moreover, a set of performance comparisons between single-agent reinforcement learning and multiagent reinforcement learning were performed in [6]. In [23], 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 [13], Kapetanakis et al. propose the technique that converge almost always in fully stochastic environments, i.e. when rewards are stochastic.

A more recent study falling into the third category is [5], in which uses a policy gradient method originally proposed by Sutton, McAllester, Singh and Mansour in [26], 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 (Win or Learn Fast) principle [4] to adjust the learning rate so that the convergence is guaranteed (albeit only with very simple problems). Another study on policy gradients is [21], in which the VAPS framework originally proposed by Baird and Moore in [1] for single-agent domains is expanded for cooperative games.

The first learning method for multistate Markov games was proposed by Littman in [16]. 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 in [10]. Unfortunately, their method is guaranteed to converge only under very restrictive conditions. Littman proposed a new method in [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 in [28]. Conitzer and Sandholm presented an algorithm that converges to a Nash equilibrium in self-play and learn to play optimally against stationary opponents in [7]. Sun and Qi propose in [24] a method for alternating turn games and study consequences of making rationality assumptions about opponents.

Another problem that arises with general-sum Markov games is the computational complexity of the calculation of Nash equilibrium solutions. It is still an open question if there exist computationally efficient methods for determining Nash equilibria in an arbitrary game. 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 in [9]. 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 [8].

A totally different approach to multiagent reinforcement learning is the COllective INtelligence (COIN) architecture proposed by Wolpert and Tumer [34]. The COIN architecture can been seen rather as a framework for adjusting single-agent reinforcement learning algorithms for multiagent domains than a standalone 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 realworld applications, see e.g. [35] and [33].

Our previous contributions in the field of multiagent reinforcement learning include an asymmetric multiagent reinforcement learning method [14], which introduces an alternative solution concept to the Nash solution concept in Markov games, i.e. the Stackelberg solution concept. Additionally, we have proposed a gradient-based learning method for both symmetric and asymmetric multiagent reinforcement learning in [15], which serves as a stepping stone for this more advanced work.

We begin the paper 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 multiagent reinforcement learning problem. Finally, we extend the VAPS for multiagent case and test it with two example problems that have reasonable large state-action spaces.

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 paper.

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

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 $r^i : A^1 \times A^2 \times \ldots \times A^N \rightarrow \mathbb{R}$ is the payoff function for player i.

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

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

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 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 to other players. For solving these kind of optimization problems we have to use a hierarchical equilibrium solution concept, i.e. the *Stackelberg equilibrium* concept. In the two-player case, where one player is acting as the leader (player 1) and the another as the follower (player 2), the leader enforces his strategy to 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.

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 Fig. 1. The environment 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.

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, $p : S \times A \rightarrow \Delta(S)$ is the state transition function and $r : S \times A \rightarrow \mathbb{R}$ is the reward

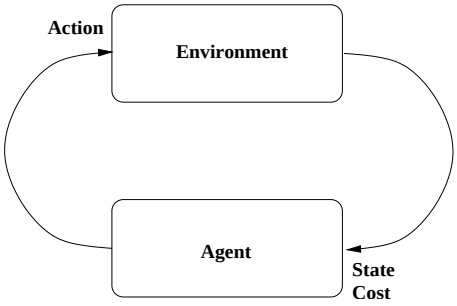

Fig. 1. An overview of the learning system.

function. $\Delta(S)$ is the set of probability distributions over the set S.

Additionally, we need a *policy*, i.e. 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:

$$\pi : S_t \rightarrow A_t, \tag{1}$$

where t refers to the discrete time step. The policy is *stationary* if there are no time dependents, i.e.

$$\pi : S \rightarrow A. \tag{2}$$

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

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

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:

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

Finding the optimal policy π_* can be seen as an optimization problem, which can be solved e.g. 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*. 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 to 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-learning* which has the update rule [22]:

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

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 [29]:

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

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, i.e. *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 $i \in \{1, N\}$, $p : S \times A^1 \times \ldots \times A^N \rightarrow \Delta(S)$ is the state transition function, $r^i : S \times A^1 \times \ldots \times A^N \rightarrow \mathbb{R}$ is the reward function for agent i. $\Delta(S)$ is the set of probability distributions over the set S.

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):

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

The expected discounted utility of agent i is the following:

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

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

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

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 Eq. (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, e.g. 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:

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

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, i.e. 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 Eq. (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 [10]:

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

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

Note that it is guaranteed that every finite matrix game possesses at least one Nash equilibrium in mixed strategies. However, there need not exist a Nash equilibrium point in pure strategies and therefore $\mathrm{Nash}\{Q^1_t(s_{t+1})\}$ in Eq. (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 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:

$$\arg \max_{\substack{a^1 \in A^1 \\ a^2 \in A^2}} Q^1_{\pi^1,\pi^2}(s, a^1, a^2). \tag{12}$$

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

$$g(s, a^c) = \arg\min_{a^1 \in A^1} \|f(Q^2_{\pi^1,\pi^2}(s, a^1, a^2)), a^c\|. \quad (13)$$

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

$$a_S^2 = \arg\max_{a^2 \in A^2} Q^2_{\pi^1,\pi^2}(s, g(s, a^c), a^2). \quad (14)$$

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 a particular action and the Q-value associated to the cooperation strategies (maximal possible payoff for the leader), i.e.

$$\|x, a^c\| = |Q^1_{\pi^1,\pi^2}(s, a^1, x) - Q^1_{\pi^1,\pi^2}(s, a^{1c}, a^{2c})|. \quad (15)$$

The function f is used to select actions by player 2; e.g. in the case of of greedy action selection $f = \arg\max_{a^2 \in A^2}$. In practical implementations of the protocol, e.g. 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.

Actual learning of the payoffs $Q^1_{\pi_S^1,\pi_S^2}$ and $Q^2_{\pi_S^1,\pi_S^2}$ can be done by using any suitable method from the field of reinforcement learning. In this paper 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:

$$Q^1_{t+1}(s_t, a_t^1, a_t^2) = (1 - \alpha_t) Q^1_t(s_t, a_t^1, a_t^2) + \alpha_t [r^1_{t+1} + \gamma \max_{b \in A^1} Q^1_t(s_{t+1}, b, Tb)] \quad (16)$$

and

$$Q^2_{t+1}(s_t, a_t^1, a_t^2) = (1 - \alpha_t) Q^2_t(s_t, a_t^1, a_t^2) + \alpha_t [r^2_{t+1} + \gamma \max_{b \in A^2} Q^2_t(s_{t+1}, g(s_{t+1}, a^c_{t+1}), b)]. \quad (17)$$

In Eq. (16), the operator Tb conducts the follower's unique response to the leader's action enforcement b.

In this paper, we use also SARSA-like on-policy versions of the above learning rules. In this case, the learning agent deviates from the equilibrium solution by using the same policy that is used for the exploration of the state-action space when it estimates the value of the state at the time instance t. Note that theoretical convergence guarantees for this kind of learning rules are not provided. However, as can be seen from the test cases at the end of the paper, this method also leads to almost as good results as off-policy learning rules. Additionally, it is more natural to use SARSA-like learning with the VAPS framework and the direct policy gradient.

5. VAPS for multiagent reinforcement learning

In this paper, we propose a class of stochastic gradient descent algorithms for multiagent reinforcement learning. This class of algorithms is called VAPS (Value And Policy Search) and was originally proposed by Baird and Moore in [1] for single-agent reinforcement learning tasks. Unlike direct update rules, VAPS takes explicitly the changing exploration policy into account. A straight consequence of this is that it is possible to combine both value function based and direct policy gradient methods in one gradient descent algorithm. In some cases, this can lead to quicker convergence than using only one of these methods alone. In this section, we extend the VAPS framework for the multiagent case with two agents. Moreover, we assume that the agents observe all action choices and rewards and know the exploration policies of both agents.

5.1. VAPS

The main idea of VAPS is to minimize the total error of the policy, i.e. minimize the error E [1]:

$$E = \sum_{t=0}^{\infty} \sum_{h_t \in H_t} P(h_t) e(h_t), \quad (18)$$

where $e(\cdot)$ is an immediate error function, H_t is the set of all possible histories of length t, $h_t \in H_t$ is a history (sequence) containing all states, actions and rewards generated by the policy until the time instance t, i.e.

$$h_t = \{s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{t-1}, a_{t-1}, r_t, s_t, a_t, r_{t+1}\}, \quad (19)$$

and $P(h_t)$ is the probability of the history h_t. Formally, $P(h_t)$ can be expressed as follows [1]:

$$P(h_t) = P(a_t|s_t) P(r_{t+1}|s_t) \times \prod_{i=0}^{t-1} P(a_i|s_i; \boldsymbol{\theta}) P(r_{i+1}|s_i) P(s_{i+1}|s_i) P(\mathrm{NE}|s_i), \quad (20)$$

where the proposition NE stands for *not end* and $\boldsymbol{\theta}$ is an arbitrary parameter vector containing parameters for the function approximating Q-values. The probability function $P(a_t|s_t)$ denotes the probability of choosing the action a_t at the time instance t as a consequence of the exploration process. The VAPS framework should be fixed to the actual learning algorithm by selecting the immediate error function $e(h_t)$ in an appropriate way. We extend this framework for the multiagent case by including terms corresponding to action choices and rewards of both agents into the history. Formally, in the multiagent case, the history h_t is as follows:

$$h_t = \{s_0, a_0^1, a_0^2, r_1^1, r_1^2, s_1, a_1^1, a_1^2, r_2^1, r_2^2, \ldots, s_{t-1}, a_{t-1}^1, a_{t-1}^2, r_t^1, r_t^2, s_t, a_t^1, a_t^2, r_{t+1}^1, r_{t+1}^2\}, \quad (21)$$

where a_t^1 and a_t^2 are the actions selected by the agents at the time instance t. Respectively, r_{t+1}^1 and r_{t+1}^2 are rewards for the agents at the time instance t.

For generality, we assume that the action selection probabilities of both agents depend on the parameter vectors $\boldsymbol{\theta}^i, i = 1, 2$. Formally, the probability of the history h_t is as follows:

$$\begin{aligned} P(h_t) &= P(a_t^1|s_t)P(a_t^2|s_t)P(r_{t+1}^1|s_t)P(r_{t+1}^2|s_t) \\ &\times \prod_{i=0}^{t-1} P(a_i^1|s_i;\boldsymbol{\theta}^1,\boldsymbol{\theta}^2)P(a_i^2|s_i;\boldsymbol{\theta}^1,\boldsymbol{\theta}^2) \quad (22) \\ &\times P(r_{i+1}^1|s_i)P(r_{i+1}^2|s_i)P(s_{i+1}|s_i)P(\mathrm{NE}|s_i). \end{aligned}$$

Moreover, in the multiagent case, we denote the expected total error of agent i as:

$$E^i = \sum_{t=0}^{\infty} \sum_{h_t \in H_t} P(h_t) e^i(h_t), \quad (23)$$

where $e^i(h_t)$ is an immediate error function for agent i. Various error functions are proposed below. By substituting Eq. (22) into Eq. (23) and differentiating with respect to an arbitrary parameter θ in the vector $\boldsymbol{\theta}^i$, we get the following equation for the gradient of E^i:

$$\begin{aligned} \frac{\partial E^i}{\partial \theta} = \sum_{t=0}^{\infty} \sum_{h_t \in H_t} P(h_t) \Bigg[& \frac{\partial e^i(h_t)}{\partial \theta} \\ & + e^i(h_t) \sum_{j=0}^{t-1} \frac{\partial}{\partial \theta} \Big(\ln P(a_j^1|s_j;\boldsymbol{\theta}^1,\boldsymbol{\theta}^2) \quad (24) \\ & + \ln P(a_j^2|s_j;\boldsymbol{\theta}^1,\boldsymbol{\theta}^2) \Big) \Bigg]. \end{aligned}$$

Note that because $P(h_t)$ is a real number, the corresponding term can be omitted from Eq. (24) and we still get an unbiased estimate of the gradient. However, this is true only if the state transitions are sampled by following the current stochastic policy.

The probability terms in Eq. (24) correspond to the exploration process. For example, if we are using the softmax action selection method with the asymmetric learning model, a possible choice for this probability distribution is the *Gibbs distribution*, i.e. in the case of agent 1:

$$P(a^1|s;\boldsymbol{\theta}^1,\boldsymbol{\theta}^2) = \frac{e^{kQ^1(s,a^1,Ta^1)}}{\sum_{a \in A^1} e^{kQ^1(s,a,Ta)}}, \quad (25)$$

where k is the temperature parameter. Note that the Gibbs distribution approaches greedy action selection when k approaches infinity and therefore we select eventually the actual Stackelberg equilibrium point. Correspondingly, for agent 2, the Gibbs distribution is as follows:

$$P(a^2|s;\boldsymbol{\theta}^1,\boldsymbol{\theta}^2) = \frac{e^{kQ^2(s,a^{1*},a^2)}}{\sum_{a \in A^2} e^{kQ^2(s,a^{1*},a)}}, \quad (26)$$

where a^{1*} is the enforcement of agent 1. Additionally, in the experiments of this paper, we make the simplifying assumption that the operator T and the equilibria do not depend on the weights $\boldsymbol{\theta}^1$ and $\boldsymbol{\theta}^2$. Due to this assumption, the VAPS works much like in the single-agent case: the behavior of the opponent is assumed to be fixed. This simplification worked fine in all test cases presented at the end of this paper.

A desirable property of the time-varying action selection function is that it would eventually approach the real equilibrium point, particularly the greedy action selection in the deterministic case. In the symmetric case, the equilibrium point can be stochastic and the action selection function $P(a_t^i|s_t)$ should be a combination of the true equilibrium probabilities and the probabilities generated by the exploration process. The Nash operator does depend on the weights $\boldsymbol{\theta}^1$ and $\boldsymbol{\theta}^2$ but is not a smooth function of these weights. This implies difficulties in the calculation of the probability terms in Eq. (24). However, if the immediate error function does not depend on the current policy, as in the Q-learning, it is possible to use e.g. greedy action selection.

In this paper we test the symmetric learning model with a Q-learning type off-policy method. We sample the opponent's action from an uniform distribution in each state and then select the action, based on the opponent's action selection, by using the Gibbs distribution. This method guarantees sufficient exploration of the state-action space; however, the exploration pol-

G2		**G1**
1		**2**

Fig. 2. The game board used in the example problem. Agents are initially located in the cells marked with numbers **1** and **2**. Goal cells are marked with symbols **G1** and **G2**.

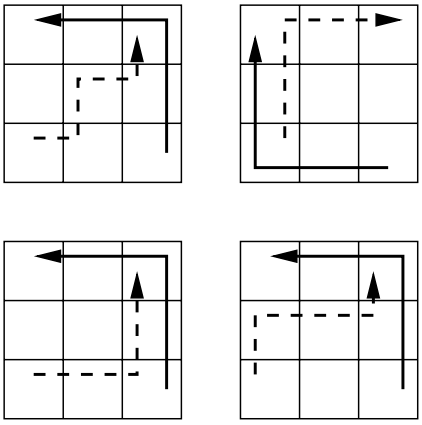

Fig. 3. Some optimal paths generated by both symmetric and asymmetric multiagent reinforcement learning models.

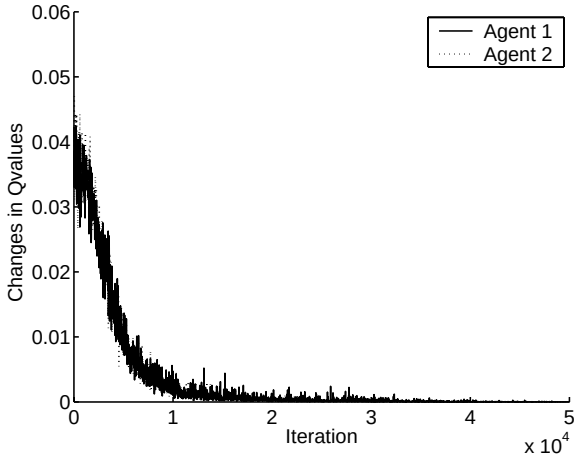

Fig. 4. Convergence of the symmetric learning model with Q-learning.

icy does not approach an equilibrium point during the learning.

If the underlying Markov game has terminal states, a natural way is to set the length of the history to the length of an *episode*, i.e. the trace from a start state to an end state. After the episode is completed (the end state is found) the history is cleared. In this case, we are minimizing the expected error of the episode.

5.2. Error functions in multiagent domains

The choice of the immediate error function $e^i(h_t)$ stipulates the actual learning algorithm. In principle, it is possible to use any learning rule that can be expressed in the time difference form with the VAPS framework. For example, in the symmetric case with Q-learning, the immediate error function takes the following form:

$$\begin{aligned} e^i(h_t)_Q = \frac{1}{2} \sum_{s_t \in S} & P(s_t|s_{t-1}, a^1_{t-1}, a^2_{t-1}) \\ & \times [r^i_t + \gamma \mathrm{Nash}\{Q^i_{t-1}(s_t)\} \\ & - Q^i_{t-1}(s_{t-1}, a^1_{t-1}, a^2_{t-1})]^2. \end{aligned} \tag{27}$$

Correspondingly, the SARSA error function in the asymmetric case takes the following form:

$$\begin{aligned} e^i(h_t)_{\mathrm{SARSA}} = \frac{1}{2} \sum_{s_t \in S} & P(s_t|s_{t-1}, a^1_{t-1}, a^2_{t-1}) \\ & \times \sum_{a^1_t \in A^1} P(a^1_t|s_t) \sum_{a^2_t \in A^2} P(a^2_t|s_t) \\ & \times [r^i_t + \gamma Q^i_{t-1}(s_t, a^1_t, a^2_t) \\ & - Q^i_{t-1}(s_{t-1}, a^1_{t-1}, a^2_{t-1})]^2. \end{aligned} \tag{28}$$

Since Eq. (24) includes the immediate error term directly, it is possible to use a combination of the value function based and direct policy gradient learning in the VAPS framework. In the case of SARSA-learning, the total immediate error function is a linear combination of the error generated by the value function approximation and the direct reward:

$$e^i(h_t) = (1-\beta) e^i(h_t)_{\mathrm{SARSA}} + \beta(b - \gamma^t r^i_{t+1}), \tag{29}$$

where r^i_{t+1} is sampled by following the current policy and $\beta \in [0, 1]$ is the weight of the direct rewards. When $\beta = 0$, VAPS performs a direct value function based learning and when $\beta = 1$, VAPS reduces to the REINFORCE algorithm proposed by Williams in [31] and [32]. The parameter b sets the reward baseline. However, there is not so much evidence on how this parameter should be adjusted. Some theoretical suggestion can be found in [30].

If the problem instance is highly non-Markov, it would be necessary to use β-value of one or very nearly equal to one. This is especially true in partially observed Markov decision problems in which the agents do not observe the actual state but some function of this state.

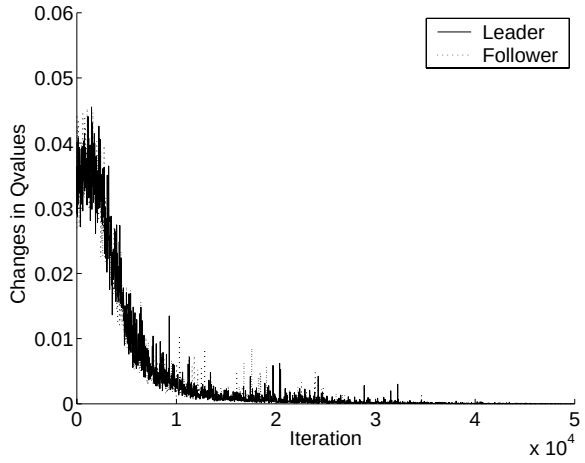

Fig. 5. Convergence of the asymmetric learning model with Q-learning.

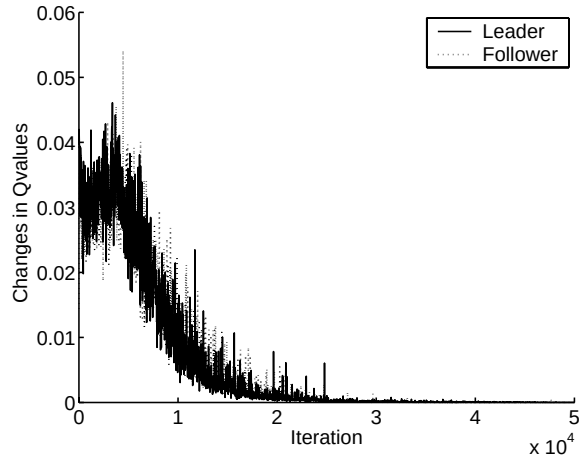

Fig. 6. Convergence of the asymmetric learning model with SARSA-learning.

5.3. Calculation of equilibrium points in matrix games

Both symmetric and asymmetric learning models require the calculation of the equilibrium point in each iteration of the learning algorithm, i.e. the calculation of an equilibrium solution of the matrix game associated with the next state. In this paper we need to calculate the Nash equilibrium point in the case of the symmetric learning model and the Stackelberg equilibrium in the case of the asymmetric learning model. The Nash equilibrium solution of the matrix game is guaranteed to exist only in mixed strategies of the players [20] and we thus need to use some sophisticated mathematical optimization algorithm for obtaining the solution. On the other hand, every matrix game has a Stackelberg equilibrium solution (not necessarily unique) in pure strategies and thus it is possible to solve this problem by simply enumerating all possible action combinations of the leader and the follower.

The calculation of the Nash equilibrium solution depends heavily on the number of players and the exact game structure. Most of the developed methods calculate the equilibrium solution in the two-player game (note that in the case of the single player, the Nash equilibrium solution is simply the maximum of the player's utility function) and the general payoff structure. There exists a number of methods for computing a sample equilibrium. It is still an open question if there exist computationally efficient methods for finding all Nash equilibria. In this work, we use the *Lemke-Howson*-algorithm for determining Nash equilibria. A good survey of methods for Nash equilibrium computation is [18].

Obtaining a mixed strategy Stackelberg equilibrium of a matrix game leads to a bilevel optimization problem that is a special case of hierarchical optimization problems. A thorough presentation of bilevel optimization problems and their actual solution procedures can be found in [2]. In this paper, we restrict our attention only to Stackelberg solutions in pure strategies.

5.4. Convergence issues

Symmetric multiagent reinforcement learning is not guaranteed to converge to optimal values. This problem is partly due to the fact that the Nash equilibrium of a matrix game is not unique and, in general, the Nash operator used with symmetric multiagent reinforcement learning in Eq. (11) returns the value of an arbitrary equilibrium. Another problem with symmetric multiagent reinforcement learning is the coordination of equilibrium selection. If agent 1 selects an other equilibrium than agent 2, the result is not a Nash equilibrium. This problem can be solved by using the same deterministic algorithm in all agents in the system.

In asymmetric multiagent reinforcement learning, a Stackelberg equilibrium is not unique. However, all Stackelberg equilibria share the same value for the leader and therefore it makes sense to select always the same equilibrium point for the leader. The only requirement is that the follower's response in unique. This problem can be circumvented by setting additional restrictions to the follower's action selection method or by adding extra goals to the leader, e.g. so that the leader does his enforcement in a risk averse manner [3].

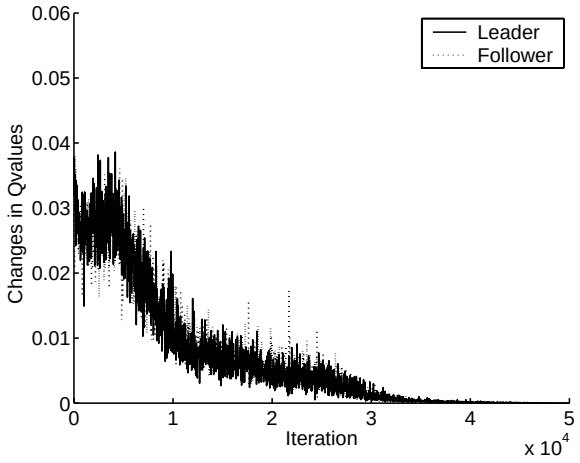

Fig. 7. Convergence of the asymmetric learning model with SARSA-policy-learning. β=0.1.

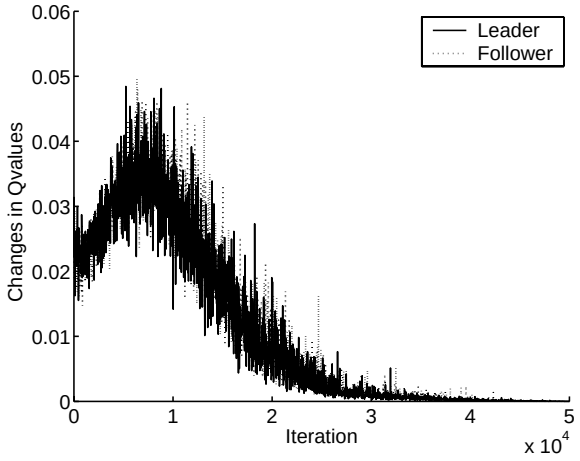

Fig. 8. Convergence of the asymmetric learning model with SARSA-policy-learning. $\beta = 0.3$.

Whether a symmetric or asymmetric learning model is applied to a problem with a general-sum payoff structure, small variations in the opponent's value function may lead to remarkable changes in the strategies and further values of the states. Therefore it is very difficult to provide exact convergence proofs for learning systems with general-sum payoff structure. However, convergent algorithms exist for team games (both agents share the same utility function) or for zero-sum games, cf. [28] and [16], respectively.

6. Example problems

In this section, we solve a simple example problem by using the VAPS framework with both symmetric and asymmetric reinforcement learning models discussed earlier in this paper. This problem is a variation of the commonly-used *grid world problem*, used for testing single-agent and multiagent reinforcement learning algorithms in many works, e.g. in [25,19,9], and [17]. Our test setting is the same as in [10], where the problem was solved by using a tabular version of the symmetric multiagent reinforcement learning algorithm. In addition, we test the VAPS framework with a simple pricing application.

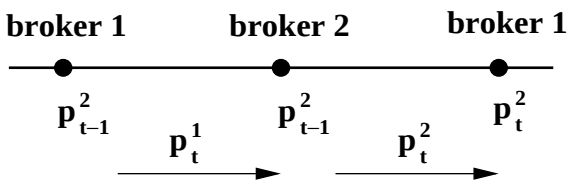

Fig. 9. Timeline of the price decisions in the pricing problem. The price symbols below the dots describe the states and the symbols above the arrows represent price decisions.

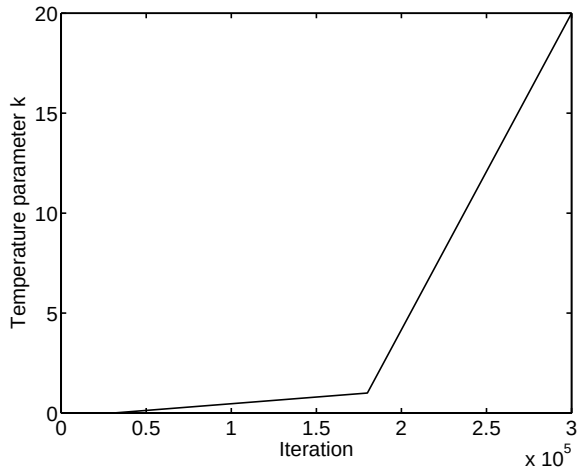

Fig. 10. Temperature parameter k plotted against the iteration number.

6.1. Grid world problem

In the grid world problem we have a grid world containing nine cells and two competing agents (Fig. 2). The initial positions of the agents are the lower corners **1** and **2**, respectively, and on each round they can move to the adjacent cells (4-neighborhood). Moreover, there are two distinct goal positions; **G1** for agent 1 and **G2** for agent 2. The agents get large positive payoffs when they reach the right goal positions. In the symmetric learning model both agents get small negative payoffs and are returned back to their original positions when

they try to move to the same position. In the asymmetric learning model, only agent 1 (leader) gets the negative payoff and is thus trying to avoid the collision using its enforcements. In all cases, the ultimate objective of the agents is to reach the goal positions using as few moves as possible.

We characterize the problem with the following Markov game:

- A state in this problem is a pair $s = (p_1, p_2)$, i.e. the positions of the agents. Hence, the state space of this example consists of $9 \times 9 = 81$ states.
- Both agents get positive payoffs of 0.9 when they find the right goal cells.
- The action set of agent i is $A^i = \{\text{Left}, \text{Right}, \text{Up}, \text{Down}\}, i = 1, 2$. The agents are restricted to stay on the game board.
- The discount factor γ is 0.99.
- Both agents select their actions using the softmax action selection method.
- In the asymmetric model agent 1 is acting as the leader.
- If the agents collide, both agents get small negative rewards of -0.1 in the symmetric model, whereas only the leader gets this negative reward in the asymmetric model.

The learning rate parameter and the temperature in the softmax action selection are modified linearly with time. The range of parameter k in Eqs (25) and (26) was [0,20]. Note that the learning rate decaying scheme does not obey the conditions defined in stochastic approximation theory. However, in real problems, using a learning rate decaying scheme that satisfies these theoretical conditions would often induce very slow convergence and such scheme is thus seldom used in applications and in empirical research.

The test runs were carried out using the off-policy (Q) learning method with the symmetric learning model and both off-policy and on-policy learning with the asymmetric learning model. In addition, test runs were repeated with the SARSA-policy algorithm ($\beta = 0.1$ and $\beta = 0.3$) in the case of the asymmetric learning model.

The value function is approximated with a linear function with one parameter for each state-actions tuple. Hence the cost-to-go function takes the following form:

$$Q(s, a^1, a^2) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s, a^1, a^2), \tag{30}$$

where $\boldsymbol{\theta}$ is a $|S||A^1||A^2|$-dimensional vector and $\boldsymbol{\phi}$ is a unit vector with the component corresponding to the state-actions tuple set to one.

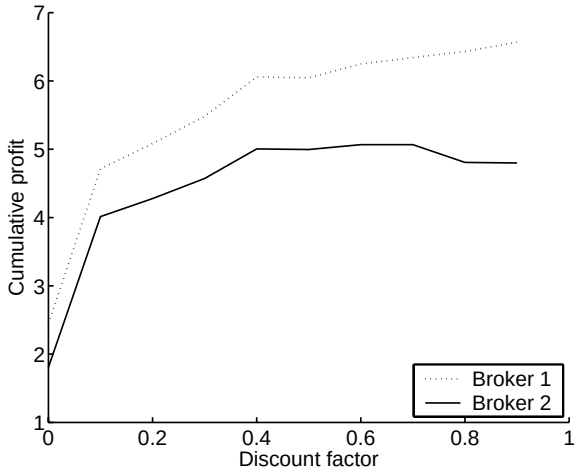

Fig. 11. Averaged profits in the pricing problem. All data points are averages of 1000 test runs each containing 10 pricing decisions for both agents.

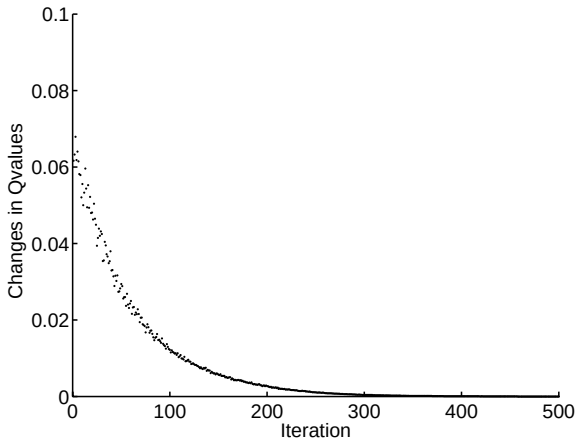

Fig. 12. Convergence of the leader's Q-values in the pricing problem. $\gamma = 0.3$.

When an agent reaches the right goal position, both agents are moved back to their initial positions and the test run is restarted with a new episode. These episodes also define the history sequence used with the VAPS framework and hence the history is cleared after each episode. The maximum length of an episode is restricted to 10 moves. A few optimal paths generated by the learning models are illustrated in Fig. 3.

We repeated the test runs 50 times and the averaged convergence curves are shown in Figs 4–8, where the Euclidean distance between vectors containing values from consecutive training rounds is plotted against the round number. Both agents converged in each test run with an equal pace. The variance of the convergence speed was higher with on-policy learning than with

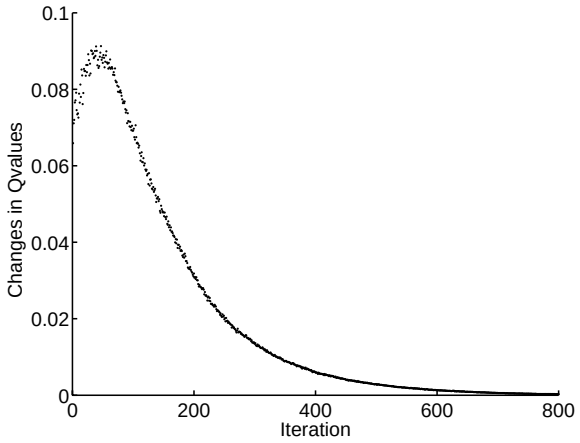

Fig. 13. Convergence of the leader's Q-values in the pricing problem. $\gamma = 0.7$.

off-policy learning. The reason is that the on-policy methods are more sensitive to the exploration policy than the off-policy methods. The overall convergence was slightly slower with the SARSA-policy method than with the normal SARSA-learning.

6.2. Pricing problem

In this problem, two competing agents (brokers) 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, i.e. the other broker's, current price. After the price has been set, the customer either buys a product from one seller or decides not to buy the product at all. The objective of the agents is to maximize their profits. We begin the discussion by modeling the interaction between the two brokers as an asymmetric multiagent reinforcement learning problem.

In [27], 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", i.e. repeated price reductions. As a result of a price war, the prices would go very low and the overall profits would be small. Tesauro and Kephart reported very good performance of their 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 and the learning system encounters serious convergence problems. In this paper, we model the system as a Markov game and solve it by using Q-learning and the VAPS framework.

A customer buys the product from the broker having the lowest price. After the customer has done his purchase decision, both brokers get their immediate profits according to the following utility functions:

$$u^1(p^1, p^2) = \begin{cases} p^1 - c \text{ if } p^1 \leqslant p^2 \\ 0 \qquad \text{otherwise} \end{cases} \tag{31}$$

and

$$u^2(p^1, p^2) = \begin{cases} p^2 - c \text{ if } p^1 > p^2 \\ 0 \qquad \text{otherwise,} \end{cases} \tag{32}$$

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

We make the assumption that the brokers do not decide their decisions simultaneously, i.e. 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 the broker 2.
- Broker 1 is acting as the leader and thus decides its price prior to the broker 2. Hence, as the state is the current price of the broker 2, the utility of the 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 the Stackelberg equilibrium point of the matrix game associated with the current state and makes its pricing decision based on this solution. After that, it announces its price decision to broker 2 who, in its turn, maximizes its utility value based on this enforcement. This process is illustrated in Fig. 9.

As in the previous problem, the Q-function was approximated with the linear function. The number of different pricing options was 15 for both agents and the learning rate parameter was decayed linearly during the learning. The state-action space was explored by using the Gibbs-distribution defined in Eqs (25) and (26). The parameter k was increased as illustrated in Fig. 10. At first the parameter had very small values ensuring sufficient exploration of the state-action space and was then increased linearly to the high value ($k = 20$) corresponding almost to greedy action selection. The training phase consisted of 300000 episodes

and each episode consisted of 10 pricing decisions for both brokers. The initial states were selected randomly.

In the testing phase, the initial state (price of the broker 2) was selected randomly and each test run consisted of 10 pricing decisions per broker. In Fig. 11, the cumulative profits (averages from 1000 test runs) of both agents are plotted against the discount factor γ. The average profit of broker 1 grows monotonically as the discount factor increases. The profit of broker 2 increases, albeit not monotonically. Moreover, even the use of small discount factor $\gamma = 0.1$, corresponding to a 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.

The convergence of the agents' Q-values ($\boldsymbol{\theta}$) are illustrated in Figs 12 and 13 (averages from 50 test runs), where the Euclidean distance between vectors containing values from consecutive training rounds (episodes) is plotted against the round number (only every hundredth sample is plotted). Two cases with discount factors 0.3 and 0.7 are plotted for the broker 1. It can be seen that the algorithm converged very fast in every case although 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 broker 2 are similar.

7. Conclusions and future research

A novel numerical method for both symmetric and asymmetric multiagent reinforcement learning are presented in this paper. The paper extends the VAPS framework for Markov games and the framework is suitable for various actual multiagent reinforcement learning methods. Additionally, it is possible to use a combination of value function approximation and direct policy gradients with the VAPS framework. The proposed methods were tested with two example problems and the results of the test runs were compared.

The methods proposed in this paper utilize parameterizations of the value function. Another approach is to parameterize the policy function directly and then learn these parameters from observations. Hence, in future research, we will study direct policy gradient methods in Markov games. Moreover, there is plenty of room for the research on efficient exploration strategies for multiagent reinforcement learning systems.

References

[1] L. Baird and A. Moore, Gradient descent for general reinforcement learning, in: *Advances in Neural Information Processing Systems* (vol. 11), Cambridge, MA, MIT Press, 1999.

[2] J.F. Bard, *Practical Bilevel Optimization–Algorithms and Applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[3] T. Basar and G.J. Olsder, *Dynamic Noncooperative Game Theory*, vol. 160, of *Mathematics in Science and Engineering*, Academic Press Inc. (London) Ltd., London, UK, 1982.

[4] M. Bowling and M.M. Veloso, Multiagent learning using a variable learning rate, *Artificial Intelligence* **136**(2) (2002).

[5] M. Bowling and M.M. Veloso, *Scalable learning in stochastic games*, in Proceedings of the AAAI-02 Workshop on Game Theoretic and Decision Theoretic Agents, Edmonton, Canada, 2002.

[6] C. Claus and C. Boutilier, *The dynamics of reinforcement learning in cooperative multiagent systems*, in Proceedings of the Fifteenth National Conference of Artificial Intelligence (AAAI-98), Madison, WI, AAAI Press, 1998.

[7] V. Conitzer and T. Sandholm, *AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents*, in Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, Morgan Kaufmann Publishers, 2003.

[8] V. Conitzer and T. Sandholm, *Complexity results about Nash equilibria*, in Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2003), Acapulco, Mexico, AAAI Press, 2003.

[9] A. Greenwald and K. Hall, *Correlated-Q learning*, in Proceedings of the Eighteenth International Conference on Machine Learning (ICML-2003), Washington, DC, AAAI Press, 2003.

[10] J. Hu and M.P. Wellman, Nash Q-learning for general-sum stochastic games, *Journal of Machine Learning Research* **4** (2003).

[11] S. Kapetanakis and D. Kudenko, *Improving on the reinforcement learning of coordination in cooperative multi-agent systems*, in Proceedings of the Second Symposium on Adaptive Agents and Multi-agent Systems (AAMAS-02), London, UK, 2002, Society for the Study of Artificial Intelligence and the Simulation of Behaviour.

[12] S. Kapetanakis and D. Kudenko, *Reinforcement learning of coordination in cooperative multi-agent systems*, in Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02), Edmonton, Alberta, Canada, AAAI Press, 2002.

[13] S. Kapetanakis, D. Kudenko and M. Strens, *Learning to coordinate using commitment sequences in cooperative multiagent-systems*, in Proceedings of the AISB-2003 Symposium on Adaptive Agents and Multi-Agent Systems, Aberystwyth, UK, 2003. Society for the Study of Artificial Intelligence and the Simulation of Behaviour.

[14] V.J. Könönen, *Asymmetric multiagent reinforcement learning*, in Proceedings of the 2003 WIC International Conference on Intelligent Agent Technology (IAT-2003), Halifax, Canada, IEEE Press, 2003.

[15] V.J. Könönen, *Gradient based method for symmetric and asymmetric multiagent reinforcement learning*, in Proceedings of the Fourth International Conference on Intelligent Data Engineering and Automated Learning (IDEAL 2003), Hong Kong, China, Springer-Verlag, 2003.

[16] M.L. Littman, *Markov games as a framework for multi-agent reinforcement learning*, in Proceedings of the Eleventh International Conference on Machine Learning (ICML-1994), New Brunswick, NJ, Morgan Kaufmann Publishers, 1994.

[17] M.L. Littman, *Friend-or-Foe Q-learning in general-sum games*, in Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williamstown, MA, Morgan Kaufmann Publishers, 2001.

[18] R.D. McKelvey and A. McLennan, Computation of equilibria in finite games, in: *Handbook of Computational Economics*, (vol. 1), H. Amman, D. Kendrick and J. Rust, eds, Elsevier Science, Amsterdam, The Netherlands, 1996.

[19] T.M. Mitchell, *Machine Learning*, McGraw-Hill, New York, NY, 1997.

[20] J.F. Nash, *Equilibrium points in N-person games*, Proceedings of National Academy of Sciences of the United States of America, 36, 1950.

[21] L. Peshkin, K.-E. Kim, N. Meuleau and L.P. Kaelbling, *Learning to cooperate via policy-search*, in Proceedings of the Sixteenth Conference on Uncertainty in Artifical Intelligence (UAI-2000), Stanford, CA, Morgan Kaufmann Publishers, 2000.

[22] G.A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*, Technical Report CUED/F-INFENG/TR166, Cambridge University, Engineering Department, 1994.

[23] S. Singh, M. Kearns and Y. Mansour, *Nash convergence of gradient dynamics in general-sum games*, in Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-2000), Stanford, CA, Morgan Kaufmann Publishers, 2000.

[24] R. Sun and D. Qi, *Rationality assumptions and optimality of co-learning*, in Proceedings of the Third Pacific Rim International Workshop on Multi-Agents (PRIMA-2000), Melbourne, Australia, Springer-Verlag, 2000.

[25] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[26] R.S. Sutton, D. McAllester, S. Singh and Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: *Advances in Neural Information Processing Systems*, (vol. 12), Cambridge, MA, MIT Press, 2000.

[27] G. Tesauro and J.O. Kephart, *Pricing in agent economies using multi-agent Q-learning*, in Proceedings of Workshop on Game Theoretic and Decision Theoretic Agents (GTDT'99), London, UK, 1999.

[28] X. Wang and T. Sandholm, Reinforcement learning to play an optimal Nash equilibrium in team Markov games, in: *Advances in Neural Information Processing Systems*, (vol. 15), Cambridge, MA, MIT Press, 2003.

[29] C.J.C.H Watkins, *Learning from Delayed Rewards*, PhD thesis, Cambridge University, Cambridge, UK, 1989.

[30] L. Weaver and N. Tao, *The optimal reward baseline for gradient-based reinforcement learning*, in Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI-2001), Seattle, WA, Morgan Kaufman Publishers, 2001.

[31] R.J. Williams, *Toward a theory of reinforcement-learning connectionist systems*, Technical Report NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA, 1988.

[32] R.J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Machine Learning* **8**(3–4) (1992).

[33] D.H. Wolpert, S. Kirshner, C.J. Merz and K. Tumer, *Adaptivity in agent-based routing for data networks*, in Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000), Barcelona, Spain, ACM Press, 2000.

[34] D.H. Wolpert and K. Tumer, *An introduction to collective intelligence*, Technical Report NASA-ARC-IC-99-63, NASA Ames Research Center, 2000.

[35] D.H. Wolpert, K. Tumer and J. Frank, Using collective intelligence to route Internet traffic, in: *Advances in Neural Information Processing Systems*, (vol. 11), Cambridge, MA, MIT Press, 1999.