Delft University of Technology

Delft Center for Systems and Control

Technical report 10-003

Multi-agent reinforcement learning: An overview*

L. Buşoniu, R. Babuška, and B. De Schutter

If you want to cite this report, please use the following reference instead:

L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," Chapter 7 in *Innovations in Multi-Agent Systems and Applications – 1* (D. Srinivasan and L.C. Jain, eds.), vol. 310 of *Studies in Computational Intelligence*, Berlin, Germany: Springer, pp. 183–221, 2010.

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.51.19 (secretary)
fax: +31-15-278.66.79
URL: `http://www.dcsc.tudelft.nl`

*This report can also be downloaded via `http://pub.deschutter.info/abs/10_003.html`

Multi-Agent Reinforcement Learning: An Overview

Lucian Buşoniu[1], Robert Babuška[2], and Bart De Schutter[3]

Abstract Multi-agent systems can be used to address problems in a variety of domains, including robotics, distributed control, telecommunications, and economics. The complexity of many tasks arising in these domains makes them difficult to solve with preprogrammed agent behaviors. The agents must instead discover a solution on their own, using learning. A significant part of the research on multi-agent learning concerns reinforcement learning techniques. This chapter reviews a representative selection of Multi-Agent Reinforcement Learning (MARL) algorithms for fully cooperative, fully competitive, and more general (neither cooperative nor competitive) tasks. The benefits and challenges of MARL are described. A central challenge in the field is the formal statement of a multi-agent learning goal; this chapter reviews the learning goals proposed in the literature. The problem domains where MARL techniques have been applied are briefly discussed. Several MARL algorithms are applied to an illustrative example involving the coordinated transportation of an object by two cooperative robots. In an outlook for the MARL field, a set of important open issues are identified, and promising research directions to address these issues are outlined.

Center for Systems and Control, Delft University of Technology, The Netherlands, i.l.busoniu@tudelft.nl · Center for Systems and Control, Delft University of Technology, The Netherlands, r.babuska@tudelft.nl · Center for Systems and Control & Marine and Transport Technology Department,
Delft University of Technology, The Netherlands, b@deschutter.info

Portions reprinted, with permission, from [20], 'A Comprehensive Survey of Multiagent Reinforcement Learning', by Lucian Buşoniu, Robert Babuška, and Bart De Schutter, IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews, vol. 38, no. 2, March 2008, pages 156–172. © 2008 IEEE.

1 Introduction

A multi-agent system is a group of autonomous, interacting entities sharing a common environment, which they perceive with sensors and upon which they act with actuators [107, 135, 139]. Multi-agent systems are finding applications in a variety of domains including robotic teams, distributed control, resource management, collaborative decision support systems, data mining, etc. [4, 33, 88, 100, 115, 125]. They may arise as the most natural way of looking at a system, or may provide an alternative perspective on systems that are originally regarded as centralized. For instance, in robotic teams the control authority is naturally distributed among the robots [115]. In resource management, while the resources could be managed by a central authority, identifying each resource with an agent may provide a helpful, distributed perspective on the system [33].

Although the agents in a multi-agent system can be endowed with behaviors designed in advance, they often need to learn new behaviors online, such that the performance of the agent or of the whole multi-agent system gradually improves [106, 115]. This is usually because the complexity of the environment makes the *a priori* design of good agent behaviors difficult or even impossible. Moreover, in an environment that changes over time, a hardwired behavior may become unappropriate.

A reinforcement learning (RL) agent learns by interacting with its dynamic environment [58, 106, 120]. At each time step, the agent perceives the state of the environment and takes an action, which causes the environment to transit into a new state. A scalar reward signal evaluates the quality of each transition, and the agent has to maximize the cumulative reward along the course of interaction. The RL feedback (the reward) is less informative than in supervised learning, where the agent would be given the correct actions to take [27] (such information is unfortunately not always available). The RL feedback is, however, more informative than in unsupervised learning, where there is no explicit feedback on the performance [104]. Well-understood, provably convergent algorithms are available for solving the single-agent RL task. Together with the simplicity and generality of the setting, this makes RL attractive also for multi-agent learning.

This chapter provides a comprehensive overview of multi-agent reinforcement learning (MARL) . We mainly focus on autonomous agents learning how to solve *dynamic* tasks online, using algorithms that originate in *temporal-difference RL* . We discuss the contributions of game theory to MARL , as well as important algorithms for static tasks.

We first outline the benefits and challenges of MARL . A central challenge in the field is the definition of an appropriate formal goal for the learning multi-agent system. We present the different learning goals proposed in the literature, which consider the stability of the agent's learning dynamics on the one hand, and its adaptation to the changing behavior of the other agents on the other hand. The core of the chapter consists of a detailed study of a representative selection of MARL algorithms, which allows us to identify the structure of the field and to provide insight into the state of the art. This study organizes the algorithms first by the type of

task they address: fully cooperative, fully competitive, and mixed (neither cooperative nor competitive); and then by the type of learning goal they target: stability, adaptation, or a combination of both. Additionally, we briefly discuss the problem domains where MARL techniques have been applied, and we illustrate the behavior of several MARL algorithms in a simulation example involving the coordinated transportation of an object by two cooperative agents. In an outlook for the MARL field, we identify a set of important open issues and suggest promising directions to address these issues.

The remainder of this chapter is organized as follows. Section 2 introduces the necessary background in single-agent RL , multi-agent RL , and game theory. Section 3 reviews the main benefits and challenges of MARL , and Section 4 presents the MARL goals proposed in the literature. In Section 5, MARL algorithms are classified and reviewed in detail. Section 6 reviews several application domains of MARL , while Section 7 provides an example involving object transportation. Finally, Section 8 distills an outlook for the MARL field, Section 9 presents related work, and Section 10 concludes the chapter.

2 Background: reinforcement learning

In this section, the necessary background on single-agent and multi-agent RL is introduced. First, the single-agent task is defined and its solution is characterized. Then, the multi-agent task is defined. Static multi-agent tasks are introduced separately, together with necessary game-theoretic concepts. The discussion is restricted to discrete state and action spaces having a finite number of elements, as a large majority of MARL results is given for this setting.

2.1 The single-agent case

The formal model of single-agent RL is the *Markov decision process*.

Definition 1. A finite Markov decision process is a tuple $\langle X, U, f, \rho \rangle$ where X is the finite set of environment states, U is the finite set of agent actions, $f : X \times U \times X \to [0,1]$ is the state transition probability function, and $\rho : X \times U \times X \to \mathbb{R}$ is the reward function.[1]

The state $x_k \in X$ describes the environment at each discrete time step k. The agent observes the state and takes an action $u_k \in U$. As a result, the environment changes its state to some $x_{k+1} \in X$ according to the transition probabilities given by

[1] Throughout the chapter, the standard control-theoretic notation is used: x for state, X for state space, u for control action, U for action space, f for environment (process) dynamics. We denote reward functions by ρ, to distinguish them from the instantaneous rewards r and the returns R. We denote agent policies by h.

f: the probability of ending up in x_{k+1} after u_k is executed in x_k is $f(x_k,u_k,x_{k+1})$. The agent receives a scalar reward $r_{k+1} \in \mathbb{R}$, according to the reward function ρ: $r_{k+1} = \rho(x_k,u_k,x_{k+1})$. This reward evaluates the immediate effect of action u_k, i.e., the transition from x_k to x_{k+1}. It says, however, nothing directly about the long-term effects of this action. We assume that the reward function is bounded.

For deterministic systems, the transition probability function f is replaced by a simpler transition function, $\overline{f} : X \times U \to X$. It follows that the reward is completely determined by the current state and action: $r_{k+1} = \overline{\rho}(x_k,u_k)$, $\overline{\rho} : X \times U \to \mathbb{R}$. Some Markov decision processes have terminal states, i.e., states that once reached, can no longer be left; all the rewards received from a terminal state are 0. In such a case, the learning process is usually separated in distinct *trials* (episodes), which are trajectories starting from some initial state and ending in a terminal state.

The behavior of the agent is described by its policy, which specifies how the agent chooses its actions given the state. The policy may be either stochastic, $h : X \times U \to [0,1]$, or deterministic, $\overline{h} : X \to U$. A policy is called stationary if it does not change over time. The agent's goal is to find a policy that maximizes, from every state x, the expected discounted return:

$$R^h(x) = \mathrm{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} \,\middle|\, x_0 = x, h \right\} \tag{1}$$

where $\gamma \in [0,1)$ is the discount factor, and the expectation is taken over the probabilistic state transitions under the policy h. The return R compactly represents the reward accumulated by the agent in the long run. Other possibilities of defining the return exist [58]. The discount factor γ can be regarded as encoding an increasing uncertainty about rewards that will be received in the future, or as a means to bound the sum which otherwise might grow unbounded.

The task of the agent is therefore to maximize its long-term performance (return), while only receiving feedback about its immediate, one-step performance (reward). One way it can achieve this is by computing the optimal state-action value function (*Q-function*). The Q-function $Q^h : X \times U \to \mathbb{R}$ gives the expected return obtained by the policy h from any state-action pair:

$$Q^h(x,u) = \mathrm{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} \,\middle|\, x_0 = x, u_0 = u, h \right\}$$

The optimal Q-function is defined as $Q^*(x,u) = \max_h Q^h(x,u)$. It satisfies the Bellman optimality equation:

$$Q^*(x,u) = \sum_{x' \in X} f(x,u,x') \left[\rho(x,u,x') + \gamma \max_{u'} Q^*(x',u') \right] \quad \forall x \in X, u \in U \tag{2}$$

This equation states that the optimal value of taking u in x is the expected immediate reward plus the expected (discounted) optimal value attainable from the next state (the expectation is explicitly written as a sum since X is finite).

Once Q^* is available, an optimal policy (i.e., one that maximizes the return) can be computed by choosing in every state an action with the largest optimal Q-value:

$$\overline{h}^*(x) = \arg\max_u Q^*(x,u) \tag{3}$$

When multiple actions attain the largest Q-value, any of them can be chosen and the policy remains optimal. In such a case, here as well as in the sequel, the 'arg' operator is interpreted as returning only one of the equally good solutions. A policy that maximizes a Q-function in this way is said to be greedy in that Q-function. So, an optimal policy can be found by first determining Q^* and then computing a greedy policy in Q^*.

A broad spectrum of single-agent RL algorithms exists, e.g., model-free methods based on the online estimation of value functions [6,89,118,120,137], model-based methods (typically called dynamic programming) [8,96], and model-learning methods that estimate a model, and then learn using model-based techniques [79, 119]. The model comprises the transition probabilities and the reward function. Many MARL algorithms are derived from a model-free algorithm called *Q-learning*[2] [137], see e.g., [17,42,49,67,69,70].

Q-learning [137] turns (2) into an iterative approximation procedure. *Q-learning* starts with an arbitrary Q-function, observes transitions $(x_k,u_k,x_{k+1},r_{k+1})$, and after each transition updates the Q-function with:

$$Q_{k+1}(x_k,u_k) = Q_k(x_k,u_k) + \alpha_k\left[r_{k+1} + \gamma\max_{u'} Q_k(x_{k+1},u') - Q_k(x_k,u_k)\right] \tag{4}$$

The term between square brackets is the temporal difference, i.e., the difference between the current estimate $Q_k(x_k,u_k)$ of the optimal Q-value of (x_k,u_k) and the updated estimate $r_{k+1}+\gamma\max_{u'} Q_k(x_{k+1},u')$. This new estimate is a sample of the right-hand side of the Bellman equation (2), applied to Q_k in the state-action pair (x_k,u_k). In this sample, x' is replaced by the observed next state x_{k+1}, and $\rho(x_k,u_k,x')$ by the observed reward r_{k+1}. The learning rate $\alpha_k \in (0,1]$ can be time-varying, and usually decreases with time.

The sequence Q_k provably converges to Q^* under the following conditions [53, 129, 137]:

- Explicit, distinct values of the Q-function are stored and updated for each state-action pair.
- The sum $\sum_{k=0}^{\infty}\alpha_k^2$ is finite, while $\sum_{k=0}^{\infty}\alpha_k$ is infinite.
- Asymptotically, all the state-action pairs are visited infinitely often.

The third requirement can be satisfied if, among others, the agent keeps trying all the actions in all the states with nonzero probabilities. This is called exploration, and can be done e.g., by choosing at each step a random action with probability $\varepsilon \in (0,1)$, and a greedy action with probability $(1-\varepsilon)$. The ε-greedy exploration procedure is obtained. The probability ε is usually decreased over time. Another

[2] Note that algorithm names are shown in italics throughout the chapter, e.g., *Q-learning*.

option is to use the Boltzmann exploration procedure, which in state x selects action u with probability:

$$h(x,u) = \frac{e^{Q(x,u)/\tau}}{\sum_{\bar{u}} e^{Q(x,\bar{u})/\tau}} \tag{5}$$

where $\tau > 0$, the temperature, controls the randomness of the exploration. When $\tau \to 0$, (5) becomes equivalent with greedy action selection (3). When $\tau \to \infty$, action selection is purely random. For $\tau \in (0,\infty)$, higher-valued actions have a greater chance of being selected than lower-valued ones.

2.2 The multi-agent case

The generalization of the Markov decision process to the multi-agent case is the *stochastic game*.

Definition 2. A stochastic game is a tuple $\langle X, U_1, \ldots, U_n, f, \rho_1, \ldots, \rho_n \rangle$ where n is the number of agents, X is the finite set of environment states, U_i, $i = 1,\ldots,n$ are the finite sets of actions available to the agents, yielding the joint action set $\boldsymbol{U} = U_1 \times \cdots \times U_n$, $f : X \times \boldsymbol{U} \times X \to [0,1]$ is the state transition probability function, and $\rho_i : X \times \boldsymbol{U} \times X \to \mathbb{R}$, $i = 1,\ldots,n$ are the reward functions of the agents.

We assume that the reward functions are bounded. In the multi-agent case, the state transitions are the result of the joint action of all the agents, $\boldsymbol{u}_k = [u_{1,k}^{\mathrm{T}}, \ldots, u_{n,k}^{\mathrm{T}}]^{\mathrm{T}}$, $\boldsymbol{u}_k \in \boldsymbol{U}$, $u_{i,k} \in U_i$ (where T denotes vector transpose). The policies $h_i : X \times U_i \to [0,1]$ form together the joint policy $\boldsymbol{h}$. Because the rewards $r_{i,k+1}$ of the agents depend on the joint action, their returns depend on the joint policy:

$$R_i^{\boldsymbol{h}}(x) = \mathrm{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{i,k+1} \,\middle|\, x_0 = x, \boldsymbol{h} \right\}$$

The Q-function of each agent depends on the joint action and on the joint policy, $Q_i^{\boldsymbol{h}} : X \times \boldsymbol{U} \to \mathbb{R}$, with $Q_i^{\boldsymbol{h}}(x,\boldsymbol{u}) = \mathrm{E}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{i,k+1} \,|\, x_0 = x, \boldsymbol{u}_0 = \boldsymbol{u}, \boldsymbol{h} \right\}$.

In fully cooperative stochastic games, the reward functions are the same for all the agents: $\rho_1 = \cdots = \rho_n$. It follows that the returns are also the same, $R_1^{\boldsymbol{h}} = \cdots = R_1^{\boldsymbol{h}}$, and all the agents have the same goal: to maximize the common return. If $n = 2$ and $\rho_1 = -\rho_2$, the two agents have opposing goals, and the stochastic game is fully competitive.[3] Mixed games are stochastic games that are neither fully cooperative nor fully competitive.

[3] Competition can also arise when more than two agents are involved. However, the literature on RL in fully-competitive games typically deals with the two-agent case only.

2.3 Static, repeated, and stage games

Many MARL algorithms are designed for static (stateless) games, or work in a stage-wise fashion, i.e., separately in the static games that arise in every state of the stochastic game. Next, we introduce some game-theoretic concepts regarding static games that are necessary to understand such algorithms [3, 39].

A *static (stateless) game* is a stochastic game with no state signal and no dynamics, i.e., $X = \emptyset$. A static game is described by a tuple $\langle U_1, \ldots, U_n, \rho_1, \ldots, \rho_n \rangle$, with the rewards depending only on the joint actions $\rho_i : \boldsymbol{U} \to \mathbb{R}$. When there are only two agents, the game is often called a bimatrix game, because the reward functions of each of the two agents can be represented as a $|U_1| \times |U_2|$ matrix with the rows corresponding to the actions of agent 1, and the columns to the actions of agent 2, where $|\cdot|$ denotes set cardinality. Fully competitive static games are also called zero-sum games, because the sum of the agents' reward matrices is a zero matrix. Mixed static games are also called general-sum games, because there is no constraint on the sum of the agents' rewards.

A *stage game* is the static game that arises in a certain state of a stochastic game. The reward functions of the stage game in state x are the Q-functions of the stochastic game projected on the joint action space, when the state is fixed at x. In general, the agents visit the same state of a stochastic game multiple times, so the stage game is a *repeated* game. In game theory, a repeated game is a static game played repeatedly by the same agents. The main difference from a one-shot game is that the agents can use some of the game iterations to gather information about the other agents' behavior or about the reward functions, and make more informed decisions thereafter.

In a static or repeated game, the policy loses the state argument and transforms into a strategy $\sigma_i : U_i \to [0, 1]$. An agent's strategy for the stage game arising in some state x of the stochastic game is the projection of its policy h_i on its action space U_i, when the state is fixed at x. MARL algorithms relying on the stage-wise approach learn strategies separately for every stage game. The agent's overall policy is then the aggregate of these strategies.

An important solution concept for static games is the Nash equilibrium. First, define the best response of agent i to a vector of opponent strategies as the strategy σ_i^* that achieves the maximum expected reward given these opponent strategies:

$$\mathrm{E}\{r_i \,|\, \sigma_1, \ldots, \sigma_i, \ldots, \sigma_n\} \leq \mathrm{E}\{r_i \,|\, \sigma_1, \ldots, \sigma_i^*, \ldots, \sigma_n\} \quad \forall \sigma_i \tag{6}$$

A Nash equilibrium is a joint strategy $[\sigma_1^*, \ldots, \sigma_n^*]^{\mathrm{T}}$ such that each individual strategy σ_i^* is a best-response to the others (see e.g., [3]). The Nash equilibrium describes a *status quo*, from which no agent can benefit by changing its strategy as long as all the other agents keep their strategies constant. Any static game has at least one (possibly stochastic) Nash equilibrium; some static games have multiple Nash equilibria. Many MARL algorithms reviewed in the sequel strive to converge to Nash equilibria.

Stochastic strategies (and consequently, stochastic policies) are of a more immediate importance in MARL than in single-agent RL , because they are necessary to express certain solution concepts, such as the Nash equilibrium described above.

3 Benefits and challenges in multi-agent reinforcement learning

In addition to benefits owing to the distributed nature of the multi-agent solution, such as the speedup made possible by parallel computation, multiple RL agents can harness new benefits from sharing experience, e.g., by communication, teaching, or imitation. Conversely, besides challenges inherited from single-agent RL , including the curse of dimensionality and the exploration-exploitation tradeoff, several new challenges arise in MARL : the difficulty of specifying a learning goal, the nonstationarity of the learning problem, and the need for coordination.

3.1 Benefits of MARL

Experience sharing can help RL agents with similar tasks learn faster and reach better performance. For instance, the agents can exchange information using communication [123], skilled agents may serve as teachers for the learner [30], or the learner may watch and imitate the skilled agents [95].

A speed-up can be realized in MARL thanks to parallel computation, when the agents exploit the decentralized structure of the task. This direction has been investigated in e.g., [21, 38, 43, 61, 62].

When one or more agents fail in a multi-agent system, the remaining agents can take over some of their tasks. This implies that MARL is inherently robust. Furthermore, by design most multi-agent systems also allow the easy insertion of new agents into the system, leading to a high degree of scalability.

Existing MARL algorithms often require some additional preconditions to theoretically guarantee and to exploit the above benefits [67, 95]. Relaxing these conditions and further improving the performance of MARL algorithms in this context is an active field of study.

3.2 Challenges in MARL

The curse of dimensionality is caused by the exponential growth of the discrete state-action space in the number of state and action variables (dimensions). Because basic RL algorithms like *Q-learning* estimate values for each possible discrete state or state-action pair, this growth leads directly to an exponential increase of their computational complexity. The complexity of MARL is exponential also in

the number of agents, because each agent adds its own variables to the joint state-action space. This makes the curse of dimensionality more severe in MARL than in single-agent RL .

Specifying a good MARL goal in the general stochastic game is a difficult challenge, because the agents' returns are correlated and cannot be maximized independently. Several types of MARL goals have been proposed in the literature, which consider stability of the agent's learning dynamics [50], adaptation to the changing behavior of the other agents [93], or both stability and adaptation [14, 16, 17, 26, 70]. A detailed analysis of this open problem is given in Section 4.

Nonstationarity arises in MARL because all the agents in the system are learning simultaneously. Each agent is therefore faced with a moving-target learning problem: the best policy changes as the other agents' policies change.

The exploration-exploitation trade-off requires online (single-agent as well as multi-agent) RL algorithms to strike a balance between the exploitation of the agent's current knowledge, and exploratory, information-gathering actions taken to improve that knowledge. For instance, the Boltzmann policy (5) is a simple way of trading off exploration with exploitation. The exploration procedure is crucial for the efficiency of RL algorithms. In MARL , further complications arise due to the presence of multiple agents. Agents explore to obtain information not only about the environment, but also about the other agents in order to adapt to their behavior. Too much exploration, however, can destabilize the other agents, thereby making the learning task more difficult for the exploring agent.

The need for coordination stems from the fact that the effect of any agent's action on the environment depends also on the actions taken by the other agents. Hence, the agents' choices of actions must be mutually consistent in order to achieve their intended effect. Coordination typically boils down to consistently breaking ties between equally good joint actions or strategies. Although coordination is typically required in cooperative settings, it may also be desirable for self-interested agents, e.g., if the lack of coordination negatively affects all the agents. Consider, as an example, that a number of countries have interconnected electricity networks, and each country's network is managed by an agent. Although each agent's primary goal is to optimize its own country's energy interests, the agents must still coordinate on the power flows between neighboring countries in order to achieve a meaningful solution [84].

4 Multi-agent reinforcement learning goal

In fully cooperative stochastic games, the common return can be jointly maximized. In other cases, however, the agents' returns are typically different and correlated, and they cannot be maximized independently. Specifying a good general MARL goal is a difficult problem.

In this section, the learning goals proposed in the literature are reviewed. These goals incorporate the *stability* of the learning dynamics of the agent on the one

hand, and the *adaptation* to the changing behavior of the other agents on the other hand. Stability essentially means the convergence to a stationary policy, whereas adaptation ensures that performance is maintained or improved as the other agents are changing their policies.

The goals typically formulate conditions for static games, in terms of strategies and rewards. Some of the goals can be extended to dynamic games by requiring that conditions are satisfied stage-wise for all the states of the dynamic game. In this case, the goals are formulated in terms of stage strategies and expected returns instead of strategies and rewards.

Convergence to equilibria is a basic stability requirement [42, 50]. It means the agents' strategies should eventually converge to a coordinated equilibrium. Nash equilibria are most frequently used. However, concerns have been raised regarding their usefulness [108]. For instance, one objection is that the link between stage-wise convergence to Nash equilibria and performance in the dynamic stochastic game is unclear.

In [16, 17], convergence is required for stability, and rationality is added as an adaptation criterion. For an algorithm to be convergent, the authors of [16, 17] require that the learner converges to a stationary strategy, given that the other agents use an algorithm from a predefined, targeted class of algorithms. Rationality is defined in [16, 17] as the requirement that the agent converges to a best-response when the other agents remain stationary. Though convergence to a Nash equilibrium is not explicitly required, it arises naturally if all the agents in the system are rational and convergent.

An alternative to rationality is the concept of no-regret, which is defined in [14] as the requirement that the agent achieves a return that is at least as good as the return of any stationary strategy, and this holds for any set of strategies of the other agents. This requirement prevents the learner from 'being exploited' by the other agents. Note that for certain types of static games, no-regret learning algorithms converge to Nash equilibria [54, 143].

Targeted optimality/compatibility/safety are adaptation requirements expressed in the form of bounds on the average reward [93]. Targeted optimality demands an average reward, against a targeted set of algorithms, which is at least the average reward of a best-response. Compatibility prescribes an average reward level in self-play, i.e., when the other agents use the learner's algorithm. Safety demands a safety-level average reward against all other algorithms. An algorithm satisfying these requirements does not necessarily converge to a stationary strategy.

Other properties of (but not necessarily requirements on) MARL algorithms can also be related to stability and adaptation. For instance, opponent-independent learning is related to stability, whereas opponent-aware learning is related to adaptation [15, 70]. An opponent-independent algorithm converges to a strategy that is part of an equilibrium solution regardless of what the other agents are doing. An opponent-aware algorithm learns models of the other agents and reacts to them using some form of best-response. Prediction and rationality as defined in [26] are related to stability and adaptation, respectively. Prediction is the agent's capability

to learn accurate models of the other agents. An agent is called rational in [26] if it maximizes its expected return given its models of the other agents.

Table 1 summarizes these requirements and properties of MARL algorithms. The names under which the authors refer to the stability and adaptation properties are given in the first two columns. Pointers to some relevant literature are provided in the last column.

Table 1 Stability and adaptation in MARL. Reproduced from [20], © 2008 IEEE.

Stability property	Adaptation property	Some relevant work
convergence	rationality	[17, 31]
convergence	no-regret	[14]
—	targeted optimality, compatibility, safety	[93, 108]
opponent-independent	opponent-aware	[15, 70]
equilibrium learning	best-response learning	[13]
prediction	rationality	[26]

4.0.1 Remarks and open issues

Stability of the learning process is needed, because the behavior of stable agents is more amenable to analysis and meaningful performance guarantees. Moreover, a stable agent reduces the nonstationarity in the learning problem of the other agents, making it easier to solve. Adaptation to the other agents is needed because their behavior is generally unpredictable. Therefore, a good MARL goal must include both components. Since 'perfect' stability and adaptation cannot be achieved simultaneously, an algorithm should guarantee bounds on both stability and adaptation measures. From a practical viewpoint, a realistic learning goal should also include bounds on the transient performance, in addition to the usual asymptotic requirements.

Convergence and rationality have been used in dynamic games in the stage-wise fashion already explained [16, 17]. No-regret has not been used in dynamic games, but it could be extended in a similar way. It is unclear how targeted optimality, compatibility, and safety could be extended.

5 Multi-agent reinforcement learning algorithms

This section first provides a taxonomy of MARL algorithms, followed by a detailed review of a representative selection of algorithms.

MARL algorithms can be classified along several dimensions, among which some, such as the task type, stem from properties of multi-agent systems in general. Others, like the awareness of the other agents, are specific to learning multi-agent

systems. The proposed classifications are illustrated using the set of algorithms selected for review. All these algorithms will be discussed separately in Sections 5.1–5.4.

The type of task considered by the learning algorithm leads to a corresponding classification of MARL techniques into those addressing fully cooperative, fully competitive, or mixed stochastic games. A significant number of algorithms are designed for static (stateless) tasks only. Figure 1 summarizes the breakdown of MARL algorithms by task type.

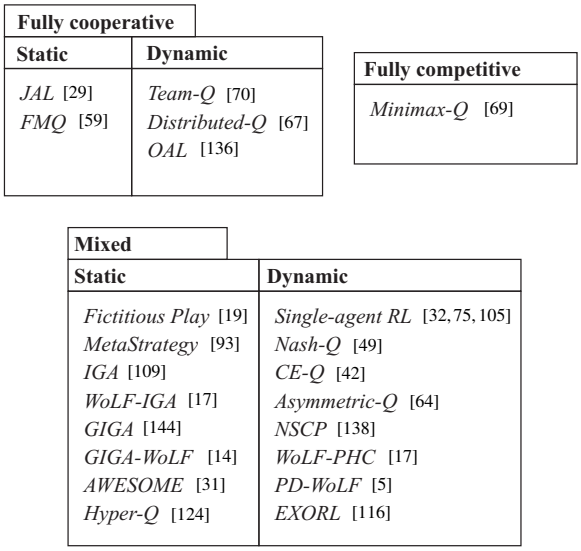


Fig. 1 Breakdown of MARL algorithms by the type of task they address. Reproduced from [20], © 2008 IEEE.

The degree of awareness of other learning agents exhibited by MARL algorithms is strongly related to the learning goal that the agents aim for. Algorithms focused on stability (convergence) only are typically unaware and *independent* of the other learning agents. Algorithms that consider adaptation to the other agents clearly need to be *aware* to some extent of their behavior. If adaptation is taken to the extreme and stability concerns are disregarded, algorithms are only *tracking* the behavior of the other agents. The degree of agent awareness exhibited by the algorithms can be determined even if they do not explicitly target stability or adaptation goals. All agent-tracking algorithms and many agent-aware algorithms use some form of opponent modeling to keep track of the other agents' policies [25, 49, 133].

The field of origin of the algorithms is a taxonomy axis that shows the variety of research inspiration contributing to MARL . MARL can be regarded as a fusion of temporal-difference RL (especially *Q-learning*), game theory, and more general direct policy search techniques. Figure 2 presents the organization of the MARL algorithms considered by their field of origin.

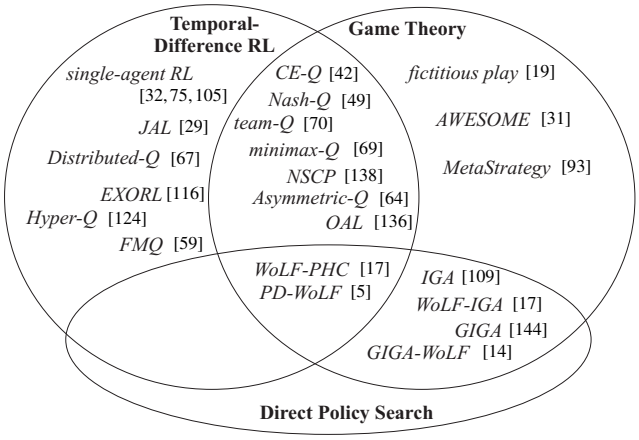


Fig. 2 MARL encompasses temporal-difference reinforcement learning, game theory and direct policy search techniques. Reproduced from [20], © 2008 IEEE.

Other classification criteria include the following:

- Homogeneity of the agents' learning algorithms: the algorithm only works if all the agents use it (homogeneous learning agents, e.g., *team-Q*, *Nash-Q*), or other agents can use other learning algorithms (heterogeneous learning agents, e.g., *AWESOME*, *WoLF-PHC*).
- Assumptions on the agent's prior knowledge about the task: a task model is available to the learning agent (model-based learning, e.g., *AWESOME*) or not (model-free learning, e.g., *team-Q*, *Nash-Q*, *WoLF-PHC*). The model consists of the transition function (unless the game is static) and of the reward functions of the agents.
- Assumptions on the agent's inputs. Typically the inputs are assumed to exactly represent the state of the environment. Differences appear in the agent's observations of other agents: it might need to observe the actions of the other agents (e.g., *team-Q*, *AWESOME*), their actions and rewards (e.g., *Nash-Q*), or neither (e.g., *WoLF-PHC*).

The remainder of this section discusses in detail the MARL algorithms selected for review. The algorithms are grouped first by the type of task they address, and then by the degree of agent awareness, as depicted in Table 2. So, algorithms for fully cooperative tasks are presented first, in Section 5.1. Explicit coordination techniques that can be applied to algorithms from any class are discussed separately in Section 5.2. Algorithms for fully competitive tasks are reviewed in Section 5.3. Finally, Section 5.4 presents algorithms for mixed tasks. Algorithms that are designed only for static tasks are discussed in separate paragraphs in the text. Simple examples are provided to illustrate several central issues that arise.

5.1 Fully cooperative tasks

In a fully cooperative stochastic game, the agents have the same reward function ($\rho_1 = \cdots = \rho_n$) and the learning goal is to maximize the common discounted return. If a centralized controller were available, the task would reduce to a Markov decision process, the action space of which would be the joint action space of the stochastic game. In this case, the goal could be achieved e.g., by learning the optimal joint-action values with *Q-learning*:

$$Q_{k+1}(x_k, \boldsymbol{u}_k) = Q_k(x_k, \boldsymbol{u}_k) + \alpha \left[r_{k+1} + \gamma \max_{\boldsymbol{u}'} Q_k(x_{k+1}, \boldsymbol{u}') - Q_k(x_k, \boldsymbol{u}_k) \right] \quad (7)$$

and then using a greedy policy. However, the agents are independent decision makers, and a coordination problem arises even if all the agents learn in parallel the common optimal Q-function using (7). It may seem that the agents could use greedy policies applied to Q^* to maximize the common return:

$$\overline{h}_i^*(x) = \arg\max_{u_i} \max_{u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_n} Q^*(x, \boldsymbol{u}) \quad (8)$$

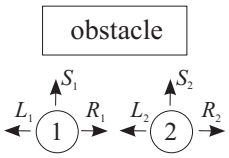
However, in certain states, multiple joint actions may be optimal. In the absence of additional coordination mechanisms, different agents may break these ties among multiple optimal joint actions in different ways, and the resulting joint action may be suboptimal.

Example 1. The need for coordination. Consider the situation illustrated in Figure 3: two mobile agents need to avoid an obstacle while maintaining formation (i.e., maintaining their relative positions). Each agent i has three available actions: go straight (S_i), left (R_i), or right (L_i).

For a given state (position of the agents), the Q-function can be projected on the joint action space. For the state represented in Figure 3 (left), a possible projection is represented in the table on the right. This table describes a fully cooperative static (stage) game. The rows correspond to the actions of agent 1, and the columns to the actions of agent 2. If both agents go left, or both go right, the obstacle is avoided while maintaining the formation: $Q(L_1, L_2) = Q(R_1, R_2) = 10$. If agent 1 goes left,

Table 2 Breakdown of MARL algorithms by task type and degree of agent awareness. Reproduced from [20], © 2008 IEEE.

Task type → ↓ Agent awareness	Cooperative	Competitive	Mixed
Independent	coordination-free	opponent-independent	agent-independent
Tracking	coordination-based	—	agent-tracking
Aware	indirect coordination	opponent-aware	agent-aware



Q	L_2	S_2	R_2
L_1	10	-5	0
S_1	-5	-10	-5
R_1	-10	-5	10

Fig. 3 Left: two mobile agents approaching an obstacle need to coordinate their action selection. Right: the common Q-values of the agents for the state depicted to the left. Reproduced from [20], © 2008 IEEE.

and agent 2 goes right, the formation is broken: $Q(L_1,R_2) = 0$. In all other cases, collisions occur and the Q-values are negative.

Note the tie between the two optimal joint actions: (L_1,L_2) and (R_1,R_2). Without a coordination mechanism, agent 1 might assume that agent 2 will take action R_2, and therefore it takes action R_1. Similarly, agent 2 might assume that agent 1 will take L_1, and consequently takes L_2. The resulting joint action (R_1,L_2) is severely suboptimal, as the agents collide.

5.1.1 Coordination-free methods

The *Team Q-learning* algorithm [70] avoids the coordination problem by assuming that the optimal joint actions are unique (which is not always the case). Then, if all the agents learn the common Q-function in parallel with (7), they can safely use (8) to select these optimal joint actions and maximize their return.

The *Distributed Q-learning* algorithm [67] solves the cooperative task without assuming coordination and with limited computation (its computational complexity is similar to that of single-agent *Q-learning*, see Section 5.4). However, the algorithm only works in deterministic problems with non-negative reward functions. Each agent i maintains a local policy $\overline{h}_i(x)$, and a local Q-function $Q_i(x,u_i)$, depending only on its own action. The local Q-values are updated only when the update leads to an increase in the Q-value:

$$Q_{i,k+1}(x_k,u_{i,k}) = \max\left\{Q_{i,k}(x_k,u_{i,k}), r_{k+1} + \gamma \max_{u'_i} Q_{i,k}(x_{k+1},u'_i)\right\} \tag{9}$$

This ensures that the local Q-value always captures the maximum of the joint-action Q-values: $Q_{i,k}(x,u_i) = \max_{u_1,\ldots,u_{i-1},u_{i+1},\ldots,u_n} Q_k(x,\boldsymbol{u})$ at all k, where $\boldsymbol{u} = [u_1,\ldots,u_n]^{\mathrm{T}}$ with u_i fixed. The local policy is updated only if the update leads to an improvement in the Q-values:

$$\overline{h}_{i,k+1}(x_k) = \begin{cases} u_{i,k} & \text{if } \max_{\bar{u}_i} Q_{i,k+1}(x_k,\bar{u}_i) > \max_{\bar{u}_i} Q_{i,k}(x_k,\bar{u}_i) \\ \overline{h}_{i,k}(x_k) & \text{otherwise} \end{cases} \tag{10}$$

This ensures that the joint policy $[\bar{h}_{1,k},\ldots,\bar{h}_{n,k}]^{\mathrm{T}}$ is always optimal with respect to the global Q_k. Under the condition that $Q_{i,0}=0\ \forall i$, the local policies of the agents provably converge to an optimal joint policy.

5.1.2 Coordination-based methods

Coordination graphs [43] simplify coordination when the global Q-function can be additively decomposed into local Q-functions that only depend on the actions of subsets of agents. For instance, in a stochastic game with 4 agents, the decomposition might be $Q(x,\boldsymbol{u})=Q_1(x,u_1,u_2)+Q_2(x,u_1,u_3)+Q_3(x,u_3,u_4)$. The decomposition might be different for different states. The local Q-functions have smaller dimensions than the global Q-function. The maximization of the joint Q-value is done by solving simpler, local maximizations in terms of the local Q-functions, and aggregating their solutions. Under certain conditions, the coordinated selection of an optimal joint action is guaranteed [43, 61].

In general, all the coordination techniques described in Section 5.2 below can be applied to fully cooperative MARL tasks. For instance, a framework to explicitly reason about possibly costly communication is the communicative multi-agent team decision problem [97].

5.1.3 Indirect coordination methods

Indirect coordination methods bias action selection toward actions that are likely to result in good rewards or returns. This steers the agents toward coordinated action selections. The likelihood of obtaining good rewards (returns) is evaluated using e.g., models of the other agents estimated by the learner, or statistics of the rewards observed in the past.

Static tasks. *Joint Action Learners (JAL)* learn joint-action values and empirical models of the other agents' strategies [29]. Agent i learns models for all the other agents $j\neq i$, using:

$$\widehat{\sigma}_j^i(u_j)=\frac{C_j^i(u_j)}{\sum_{\bar{u}_j\in U_j}C_j^i(\bar{u}_j)} \tag{11}$$

where $\widehat{\sigma}_j^i$ is agent i's model of agent j's strategy and $C_j^i(u_j)$ counts the number of times agent i observed agent j taking action u_j. Note that agent i has to observe the actions taken by the other agents. Several heuristics are proposed in [29] to increase the learner's Q-values for the actions with high likelihood of getting good rewards given the models.

The *Frequency Maximum Q-value (FMQ)* heuristic is based on the frequency with which actions yielded good rewards in the past [59]. Agent i uses Boltzmann action selection (5), plugging in modified Q-values $\tilde{Q}_i$ computed with the formula:

$$\bar{Q}_i(u_i) = Q_i(u_i) + \nu \frac{C^i_{\max}(u_i)}{C^i(u_i)} r_{\max}(u_i) \tag{12}$$

where $r_{\max}(u_i)$ is the maximum reward observed after taking action u_i, $C^i_{\max}(u_i)$ counts how many times this reward has been observed, $C^i(u_i)$ counts how many times u_i has been taken, and ν is a weighting factor. Increasing the Q-values of actions that frequently produced good rewards in the past steers the agent toward coordination. Compared to single-agent *Q-learning*, the only additional computational demands of *FMQ* come from maintaining and using the counters. However, *FMQ* can fail in some problems with strongly stochastic rewards [59], and the weighting parameter ν must be tuned in a problem-specific fashion, which may be difficult to do.

Dynamic tasks. In *Optimal Adaptive Learning (OAL)*, virtual games are constructed on top of each stage game of the stochastic game [136]. In these virtual games, optimal joint actions are rewarded with 1, and the rest of the joint actions with 0. An algorithm is introduced that, by biasing the agent towards recently selected optimal actions, guarantees convergence to a coordinated optimal joint action for the virtual game, and therefore to a coordinated joint action for the original stage game. Thus, *OAL* provably converges to optimal joint policies in any fully cooperative stochastic game. This however comes at the cost of increased complexity: each agent estimates empirically a model of the stochastic game, virtual games for each stage game, models of the other agents, and an optimal value function for the stochastic game.

5.1.4 Remarks and open issues

All the methods presented above rely on exact measurements of the state. Some of them also require exact measurements of the other agents' actions. This is most obvious for coordination-free methods: if at any point the perceptions of the agents differ, this may lead different agents to update their Q-functions differently, and the consistency of the Q-functions and policies can no longer be guaranteed.

The algorithms discussed above also suffer from the curse of dimensionality. *Distributed Q-learning* and *FMQ* are exceptions in the sense that they do not need to take into account the other agents' actions (but they only work in restricted settings: *Distributed Q-learning* only in deterministic tasks, and *FMQ* only in static tasks).

5.2 Explicit coordination mechanisms

A general approach to solving the coordination problem is to make sure that any ties are broken by all the agents in the same way, using explicit coordination or negotiation. Mechanisms for doing so based on social conventions, roles, and com-

munication, are described next [135]. These mechanisms can be used for any type of task.

Both social conventions and roles restrict the action choices of the agents. An agent role restricts the set of actions available to that agent prior to action selection, as in e.g., [112]. This means that some or all of the ties in (8) are prevented. Social conventions encode *a priori* preferences toward certain joint actions, and help break ties during action selection. If properly designed, roles or social conventions eliminate ties completely. A simple social convention relies on a unique ordering of the agents and actions [11]. These two orderings must be known to all the agents. Combining them leads to a unique ordering of the joint actions, and coordination is ensured if in (8) the first joint action in this ordering is selected by all the agents.

Communication can be used to negotiate action choices, either alone or in combination with the above coordination techniques, as in [37, 135]. When combined with the above techniques, communication can relax their assumptions and simplify their application. For instance, in social conventions, if only an ordering between agents is known, they can select actions in turn, in that order, and broadcast their selection to the remaining agents. This is sufficient to ensure coordination.

Besides action choices, agents can also communicate various other types of information, including partial or complete Q-tables, state measurements, rewards, learning parameters, etc. For example, the requirements of complete and consistent perception among all the agents (discussed under Remarks in Section 5.1) can be relaxed by allowing agents to communicate interesting data (e.g., partial state measurements) instead of relying on direct measurement [123].

Learning coordination approaches have also been investigated, where the coordination mechanism is learned, rather than being hardwired into the agents. The agents learn social conventions in [11], role assignments in [81], and the structure of the coordination graph (see Section 5.1) together with the local Q-functions in [60].

Example 2. Coordination using social conventions in a fully-cooperative task. In Example 1 above (see Figure 3), suppose the agents are ordered such that agent 1 $<$ agent 2 ($a < b$ means that a precedes b in the chosen ordering), and the actions of both agents are ordered in the following way: $L_i < R_i < S_i$, $i \in \{1,2\}$. To coordinate, the first agent in the ordering of the agents, agent 1, looks for an optimal joint action such that its action component is the first in the ordering of its actions: (L_1, L_2). It then selects its component of this joint action, L_1. As agent 2 knows the orderings, it can infer this decision, and appropriately selects L_2 in response. If agent 2 would still face a tie (e.g., if (L_1, L_2) and (L_1, S_2) would both be optimal), it could break this tie by using the ordering of its own actions (which because $L_2 < S_2$ would also yield (L_1, L_2)).

If communication is available, only the ordering of the agents has to be known. Agent 1, the first in the ordering, chooses an action by breaking ties in some way between the optimal joint actions. Suppose it settles on (R_1, R_2), and therefore selects R_1. It then communicates this selection to agent 2, which can then select an appropriate response, namely the action R_2.

5.3 Fully competitive tasks

In a fully competitive stochastic game (for two agents, when $\rho_1 = -\rho_2$), the minimax principle can be applied: maximize one's benefit under the worst-case assumption that the opponent will always endeavor to minimize it. This principle suggests using opponent-independent algorithms.

The *minimax-Q* algorithm [69, 70] employs the minimax principle to compute strategies and values for the stage games, and a temporal-difference rule similar to *Q-learning* to propagate the values across state transitions. The algorithm is given here for agent 1:

$$h_{1,k}(x_k, \cdot) = \arg \mathbf{m}_1(Q_k, x_k) \tag{13}$$

$$Q_{k+1}(x_k, u_{1,k}, u_{2,k}) = Q_k(x_k, u_{1,k}, u_{2,k}) + \alpha \left[r_{k+1} + \gamma \mathbf{m}_1(Q_k, x_{k+1}) - Q_k(x_k, u_{1,k}, u_{2,k}) \right] \tag{14}$$

where $\mathbf{m}_1$ is the minimax return of agent 1:

$$\mathbf{m}_1(Q, x) = \max_{h_1(x,\cdot)} \min_{u_2} \sum_{u_1} h_1(x, u_1) Q(x, u_1, u_2) \tag{15}$$

The stochastic strategy of agent 1 in state x at time k is denoted by $h_{1,k}(x, \cdot)$, with the dot standing for the action argument. The optimization problem in (15) can be solved by linear programming [82]. The Q-table is not subscripted by the agent index, because the equations make the implicit assumption that $Q = Q_1 = -Q_2$; this follows from $\rho_1 = -\rho_2$.

Minimax-Q is truly opponent-independent, because even if the minimax optimization has multiple solutions (strategies), any of them will achieve at least the minimax return regardless of what the opponent is doing. However, if the opponent is suboptimal (i.e., does not always take the action that is the worst for the learner), and the learner has a model of the opponent's policy, it can actually do better than the minimax return (15). An opponent model can be learned using e.g., the M^* algorithm described in [25], or a simple extension of (11) to multiple states:

$$\widehat{h}^i_j(x, u_j) = \frac{C^i_j(x, u_j)}{\sum_{\bar{u}_j \in U_j} C^i_j(x, \bar{u}_j)} \tag{16}$$

where $C^i_j(x, u_j)$ counts the number of times agent i observed agent j taking action u_j in state x.

Such an algorithm then becomes opponent-aware. Even agent-aware algorithms for mixed tasks (see Section 5.4.4) can be used to exploit a suboptimal opponent. For instance, in [17] *WoLF-PHC* was used with promising results in a fully competitive task.

Example 3. The minimax principle. Consider the situation represented in the left part of Figure 4: agent 1 has to reach the goal in the middle while still avoiding capture by its opponent, agent 2. Agent 2 on the other hand, has to prevent agent 1

from reaching the goal, preferably by capturing it. The agents can only move to the left or to the right.

Q	L_2	R_2
L_1	0	1
R_1	-10	10

Fig. 4 Left: an agent (○) attempting to reach a goal (×) while avoiding capture by another agent (●). Right: the Q-values of agent 1 for the state depicted to the left ($Q_2 = -Q_1$). Reproduced from [20], © 2008 IEEE.

For this situation (state), a possible projection of agent 1's Q-function on the joint action space is given in the table on the right. This represents a zero-sum static game involving the two agents. If agent 1 moves left and agent 2 does likewise, agent 1 escapes capture, $Q_1(L_1, L_2) = 0$; furthermore, if at the same time agent 2 moves right, the chances of capture decrease, $Q_1(L_1, R_2) = 1$. If agent 1 moves right and agent 2 moves left, agent 1 is captured, $Q_1(R_1, L_2) = -10$; however, if agent 2 happens to move right, agent 1 achieves the goal, $Q_1(R_1, R_2) = 10$. As agent 2's interests are opposite to those of agent 1, the Q-function of agent 2 is $-Q_1$. For instance, when both agents move right, agent 1 reaches the goal and agent 2 is punished with a Q-value of -10.

The minimax solution for agent 1 in this case is to move left, because for L_1, regardless of what agent 2 is doing, it can expect a return of at least 0, as opposed to -10 for R_1. Indeed, if agent 2 plays well, it will move left to protect the goal. However, it might *not* play well and move right instead. If this is true and agent 1 can find it out (e.g., by learning a model of agent 2) it can take advantage of this knowledge by moving right and achieving the goal.

5.4 Mixed tasks

In mixed stochastic games, no constraints are imposed on the reward functions of the agents. This model is most for self-interested (but not necessarily competing) agents. The influence of game-theoretic equilibrium concepts is the strongest in MARL algorithms for mixed stochastic games. When multiple equilibria exist in a particular state of a stochastic game, the equilibrium selection problem arises: the agents need to consistently pick their part of the same equilibrium.

A significant number of algorithms in this category are designed only for static tasks (i.e., repeated, general-sum games). Even in repeated games, the learning problem is still nonstationary due to the dynamic behavior of the agents playing the repeated game. This is why most of the methods in this category focus on adaptation to the other agents.

Besides agent-independent, agent-tracking, and agent-aware techniques, the application of single-agent RL methods to multi-agent learning is also presented here.

That is because single-agent RL methods do not make any assumption on the type of task, and are therefore applicable to general stochastic games, although without guarantees of success.

5.4.1 Single-agent RL

Single-agent RL algorithms like *Q-learning* can be directly applied to the multi-agent case [105]. They learn Q-functions that only depend on the current agent's action, using the basic *Q-learning* update (4), and without being aware of the other agents. The nonstationarity of the MARL problem invalidates most of the single-agent RL theoretical guarantees. Despite its limitations, this approach has found a significant number of applications, mainly because of its simplicity [32, 73–75].

One important step forward in understanding how single-agent RL works in multi-agent tasks was made in [130]. The authors of [130] applied results in evolutionary game theory to analyze the learning dynamics of *Q-learning* with Boltzmann policies (5) in repeated games. It appeared that for certain parameter settings, *Q-learning* is able to converge to a coordinated equilibrium in particular games. In other cases, unfortunately, Q-learners exhibit non-stationary cyclic behavior.

5.4.2 Agent-independent methods

Many algorithms that are independent of the other agents share a common structure based on *Q-learning*, where policies and state values are computed with game-theoretic solvers for the stage games arising in the states of the stochastic game [13, 42]. This structure is similar to that of (13)–(14); the difference is that for mixed games, solvers are usually different from minimax.

Denoting by $\left\{Q_{\cdot,k}(x,\cdot)\right\}$ the stage game arising in state x and given by all the agents' Q-functions at time k, learning takes place according to:

$$h_{i,k}(x,\cdot) = \mathbf{solve}_i\left\{Q_{\cdot,k}(x_k,\cdot)\right\} \tag{17}$$

$$Q_{i,k+1}(x_k,\boldsymbol{u}_k) = Q_{i,k}(x_k,\boldsymbol{u}_k) + \alpha\left[r_{i,k+1} + \gamma\cdot\mathbf{eval}_i\left\{Q_{\cdot,k}(x_{k+1},\cdot)\right\} - Q_{i,k}(x_k,\boldsymbol{u}_k)\right] \tag{18}$$

where $\mathbf{solve}_i$ returns agent i's part of some type of equilibrium (a strategy), and $\mathbf{eval}_i$ gives the agent's expected return given this equilibrium. The goal is to converge to an equilibrium in every state.

The updates use the Q-tables of all the agents. So, each agent needs to replicate the Q-tables of the other agents. It can do that by applying (18). This requires that all the agents use the same algorithm and can measure all the actions and rewards. Even under these assumptions, the updates (18) are only guaranteed to maintain identical results for all the agents if **solve** returns consistent equilibrium strategies for all the agents. This means the equilibrium selection problem arises when the solution of **solve** is not unique.

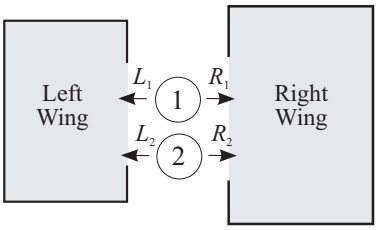
A particular instance of **solve** and **eval** for *Nash Q-learning* [49, 50] is:

$$\begin{cases} \mathbf{eval}_i\left\{Q_{\cdot,k}(x,\cdot)\right\} = V_i(x,\mathbf{NE}\left\{Q_{\cdot,k}(x,\cdot)\right\}) \\ \mathbf{solve}_i\left\{Q_{\cdot,k}(x,\cdot)\right\} = \mathbf{NE}_i\left\{Q_{\cdot,k}(x,\cdot)\right\} \end{cases} \tag{19}$$

where **NE** computes a Nash equilibrium (a set of strategies), $\mathbf{NE}_i$ is agent i's strategy component of this equilibrium, and $V_i(x,\mathbf{NE}\left\{Q_{\cdot,k}(x,\cdot)\right\})$ is the expected return for agent i from x under this equilibrium. The algorithm provably converges to Nash equilibria for all the states if either: (a) every stage game encountered by the agents during learning has a Nash equilibrium under which the expected return of all the agents is maximal; or (b) every stage game has a Nash equilibrium that is a saddle point, i.e., not only does the learner not benefit from deviating from this equilibrium, but the other agents do benefit from this [12, 49]. This requirement is satisfied only in a small class of problems. In all other cases, some external mechanism for equilibrium selection is needed to guarantee convergence.

Instantiations for *correlated equilibrium Q-learning (CE-Q)* [42] or *asymmetric Q-learning* [64] can be performed in a similar fashion, by using correlated or Stackelberg (leader-follower) equilibria, respectively. For *asymmetric Q-learning*, the follower does not need to model the leader's Q-table; however, the leader must know how the follower chooses its actions.

Example 4. The equilibrium selection problem. Consider the situation illustrated in Figure 5, left: two cleaning robots (the agents) have arrived at a junction in a building, and each needs to decide which of the two wings of the building it will clean. It is inefficient if both agents clean the same wing, and both agents prefer to clean the left wing because it is smaller, and therefore requires less time and energy.



Q_1	L_2	R_2
L_1	0	3
R_1	2	0

Q_2	L_2	R_2
L_1	0	2
R_1	3	0

Fig. 5 Left: two cleaning robots negotiating their assignment to different wings of a building. Both robots prefer to clean the smaller left wing. Right: the Q-values of the two robots for the state depicted to the left. Reproduced from [20], © 2008 IEEE.

For this situation (state), possible projections of the agents' Q-functions on the joint action space are given in the tables on the right. These tables represent a general-sum static game involving the two agents. If both agents choose the same wing, they will not clean the building efficiently, $Q_1(L_1,L_2) = Q_1(R_1,R_2) = Q_2(L_1,L_2) = Q_2(R_1,R_2) = 0$. If agent 1 takes the (preferred) left wing and agent 2 the right wing, $Q_1(L_1,R_2) = 3$, and $Q_2(L_1,R_2) = 2$. If they choose the other way around, $Q_1(R_1,L_2) = 2$, and $Q_2(R_1,L_2) = 3$.

For these returns, there are two deterministic Nash equilibria[4]: (L_1, R_2) and (R_1, L_2). This is easy to see: if either agent unilaterally deviates from these joint actions, it can expect a (bad) return of 0. If the agents break the tie between these two equilibria independently, they might do so inconsistently and arrive at a sub-optimal joint action. This is the equilibrium selection problem, corresponding to the coordination problem that arises in fully cooperative tasks. Its solution requires additional coordination mechanisms, e.g., social conventions.

5.4.3 Agent-tracking methods

Agent-tracking algorithms estimate models of the other agents' strategies or policies (depending on whether static or dynamic games are considered) and act using some form of best-response to these models. Convergence to stationary strategies is not a requirement. Each agent is assumed capable to observe the other agents' actions.

Static tasks. In the *fictitious play* algorithm [19], agent i acts at each iteration according to a best-response (6) to models $\widehat{\sigma}_1^i, \ldots, \widehat{\sigma}_{i-1}^i, \widehat{\sigma}_{i+1}^i, \ldots, \widehat{\sigma}_n^i$ learned with(11). Fictitious play converges to a Nash equilibrium in certain restricted classes of games, among which are fully cooperative, repeated games [29].

The *MetaStrategy* algorithm, introduced in [93], combines modified versions of fictitious play, minimax and a game-theoretic strategy called Bully [71] to achieve the targeted optimality, compatibility, and safety goals (see Section 4).

To compute best-responses, the *fictitious play* and *MetaStrategy* algorithms require a model of the static task, in the form of reward functions.

The *Hyper-Q* algorithm uses the other agents' models as a state vector and learns a Q-function $Q_i(\widehat{\sigma}_1, \ldots, \widehat{\sigma}_{i-1}, \widehat{\sigma}_{i+1}, \ldots, \widehat{\sigma}_n, u_i)$ with an update rule similar to *Q-learning* [124]. By learning values of strategies instead of only actions, *Hyper-Q* should be able to adapt better to nonstationary agents. One inherent difficulty is that the action selection probabilities in the models increase the dimensionality of the state space and therefore the severity of the curse of dimensionality. Additionally, the probabilities are continuous variables, which means that the classical, discrete-state *Q-learning* algorithm cannot be used. Approximate versions of *Q-learning* are required instead.

Dynamic tasks. The *Non-Stationary Converging Policies (NSCP)* algorithm [138] computes a best-response to the models and uses it to estimate state values. This algorithm is very similar to (13)–(14) and (17)–(18); this time, the stage game solver gives a best-response:

[4] There is also a stochastic (mixed) Nash equilibrium, where each agent goes left with probability $3/5$. This is because the strategies $\sigma_1(L_1) = 3/5, \sigma_1(R_1) = 2/5$ and $\sigma_2(L_2) = 3/5, \sigma_2(R_2) = 2/5$ are best-responses to one another. The expected return of this equilibrium for both agents is $6/5$, worse than for any of the two deterministic equilibria.

$$h_{i,k}(x_k,\cdot) = \arg \mathbf{br}_i(Q_{i,k},x_k) \tag{20}$$

$$Q_{i,k+1}(x_k,\boldsymbol{u}_k) = Q_k(x_k,\boldsymbol{u}_k) + \alpha\left[r_{i,k+1} + \gamma \mathbf{br}_i(Q_{i,k},x_{k+1}) - Q_k(x_k,\boldsymbol{u}_k)\right] \tag{21}$$

where the best-response value operator **br** is implemented as:

$$\mathbf{br}_i(Q_i,x) = \max_{h_i(x,\cdot)} \sum_{u_1,\ldots,u_n} h_i(x,u_i)\cdot Q_i(x,u_1,\ldots,u_n) \prod_{j=1,j\neq i}^{n} \widehat{h}_j^i(x,u_j) \tag{22}$$

The empirical models $\widehat{h}_j^i$ are learned using (16). In the computation of **br**, the value of each joint action is weighted by the estimated probability of that action being selected, given the models of the other agents (the product term in (22)).

5.4.4 Agent-aware methods

Agent-aware algorithms target convergence, as well as adaptation to the other agents. Some algorithms provably converge for particular types of tasks (mostly static), others use heuristics for which convergence is not guaranteed.

Static tasks. The algorithms presented here assume the availability of a model of the static task, in the form of reward functions. The *AWESOME* algorithm [31] uses fictitious play, but monitors the other agents and, when it concludes that they are nonstationary, switches from the best-response in fictitious play to a centrally precomputed Nash equilibrium (hence the name: Adapt When Everyone is Stationary, Otherwise Move to Equilibrium). In repeated games, *AWESOME* is provably rational and convergent [31] according to the definitions from [16,17] given in Section 4.

Some methods in the area of direct policy search use gradient update rules that guarantee convergence in specific classes of static games: *Infinitesimal Gradient Ascent (IGA)* [109], *Win-or-Learn-Fast IGA (WoLF-IGA)* [17], *Generalized IGA (GIGA)* [144], and *GIGA-WoLF* [14]. For instance, *IGA* and *WoLF-IGA* work in two-agent, two-action games, and use similar gradient update rules:

$$\begin{cases} \alpha_{k+1} = \alpha_k + \delta_{1,k}\dfrac{\partial \mathrm{E}\{r_1 \mid \alpha,\beta\}}{\partial \alpha} \\ \beta_{k+1} = \beta_k + \delta_{2,k}\dfrac{\partial \mathrm{E}\{r_2 \mid \alpha,\beta\}}{\partial \beta} \end{cases} \tag{23}$$

The strategies of the agents are represented by the probability of selecting the first out of the two actions, denoted by α for agent 1 and by β for agent 2. *IGA* uses constant gradient steps $\delta_{1,k} = \delta_{2,k} = \delta$. For an infinitesimal step size, i.e., when $\delta \to 0$, the average rewards achieved by the *IGA* policies converge to Nash rewards. In *WoLF-IGA*, $\delta_{i,k}$ switches between a smaller value when agent i is winning, and a larger value when it is losing (hence the name, Win-or-Learn-Fast). *WoLF-IGA* is rational by the definition in Section 4, and convergent for an asymptotically infinitesimal step size [17] (i.e., if $\delta_{i,k} \to 0$ when $k \to \infty$).

Dynamic tasks. *Win-or-Learn-Fast Policy Hill-Climbing (WoLF-PHC)* [17] is a heuristic algorithm that updates Q-functions with the *Q-learning* rule (4), and policies with a WoLF rule inspired from (23):

$$h_{i,k+1}(x_k,u_i) = h_{i,k}(x_k,u_i) + \begin{cases} \sum_{\bar{u}_i \neq u_i} \delta_{i,k}^{\bar{u}_i} & \text{if } u_i = \arg\max_{u_i'} Q_{i,k+1}(x_k,u_i') \\ -\delta_{i,k}^{u_i} & \text{otherwise} \end{cases} \tag{24}$$

$$\text{where } \delta_{i,k}^{u_i} = \min\left\{ h_{i,k}(x_k,u_i), \frac{\delta_{i,k}}{|U_i|-1} \right\} \tag{25}$$

$$\text{and } \delta_{i,k} = \begin{cases} \delta_{\text{win}} & \text{if winning} \\ \delta_{\text{lose}} & \text{if losing} \end{cases} \tag{26}$$

The probability decrements for the sub-optimal actions are bounded in (25) to ensure that all the probabilities remain non-negative, while the probability increment for the optimal action in (24) is chosen so that the probability distribution remains valid. The gradient step $\delta_{i,k}$ is larger when agent i is losing than when it is winning: $\delta_{\text{lose}} > \delta_{\text{win}}$. For instance, in [17] δ_{lose} is 2 to 4 times larger than δ_{win}. The rationale is that the agent should escape quickly from losing situations, while adapting cautiously when it is winning, in order to encourage convergence. The win/lose criterion in (26) is based on a comparison of an average policy with the current one in the original version of *WoLF-PHC*, and on the second-order difference of policy elements in *PD-WoLF* [5].

The *Extended Optimal Response (EXORL)* heuristic [116] applies a complementary idea in two-agent tasks: the policy update is biased in a way that minimizes the other agent's incentive to deviate from its current policy. Thus, convergence to a coordinated Nash equilibrium is encouraged.

5.4.5 Remarks and open issues

Static, repeated games represent a limited set of applications. Algorithms for static games provide valuable theoretical insight; these algorithms should however be extended to dynamic stochastic games in order to become interesting for more general classes of applications (e.g., *WoLF-PHC* [17] is such an extension). Many algorithms for mixed stochastic games, especially agent-independent algorithms, are sensitive to imperfect observations.

Game theory induces a bias toward static, stage-wise solutions in the dynamic case, as seen e.g., in the agent-independent *Q-learning* template (17)–(18). However, the suitability of such state-wise solutions in the context of the dynamic task is not always clear [86, 108].

One important research direction is understanding the conditions under which single-agent RL works in mixed stochastic games, especially in light of the preference towards single-agent techniques in practice. This direction was pioneered by the analysis in [130].

6 Application domains

MARL has been applied to a variety of problem domains, mostly in simulation but also in some real-life tasks. Simulated domains dominate for two reasons. The first reason it is easier to understand and to derive insight from results in simpler domains. The second reason is that scalability and robustness to imperfect observations are necessary in real-life tasks, and few MARL algorithms exhibit these properties. In real-life applications, more direct derivations of single-agent RL (see Section 5.4.1) are preferred [73–75, 113].

In this section, several representative application domains are reviewed: distributed control, multi-robot teams, trading agents, and resource management.

6.1 Distributed control

In distributed control, a set of autonomous, interacting controllers act in parallel on the same process. Distributed control is a meta-application for cooperative multi-agent systems: any cooperative multi-agent system is a distributed control system where the agents are the controllers, and their environment is the controlled process. For instance, in cooperative robotic teams the control algorithms of the robots identify with the controllers, and the robots' environment together with their sensors and actuators identify with the process.

Particular distributed control domains where MARL is applied are process control [113], control of traffic signals [4, 141], and control of electrical power networks [100].

6.2 Robotic teams

Robotic teams (also called multi-robot systems) are the most popular application domain of MARL , encountered under the broadest range of variations. This is mainly because robotic teams are a very natural domain for multi-agent systems, but also because many MARL researchers are active in the robotics field. The robots' environment is a real or simulated spatial domain, most often having two dimensions. Robots use MARL to acquire a wide spectrum of skills, ranging from basic behaviors like navigation to complex behaviors like playing soccer.

In navigation, each robot has to find its way to a fixed or changing goal position, while avoiding obstacles and harmful interference with other robots [17, 50].

Area sweeping involves navigation through the environment for one of several purposes: retrieval of objects, coverage of as much environment surface as possible, and exploration, for which the robots have to bring into sensor range as much of the environment surface as possible [73–75].

Multi-target observation is an extension of the exploration task, where the robots have to maintain a group of moving targets within sensor range [35, 128].

Pursuit involves the capture of moving targets by the robotic team. In a popular variant, several 'predator' robots have to capture a 'prey' robot by converging on it [52, 60].

Object transportation requires the relocation of a set of objects into a desired final configuration. The mass or size of some of the objects may exceed the transportation capabilities of one robot, thus requiring several robots to coordinate in order to bring about the objective [74]. Our example in Section 7 belongs to this category.

Robot soccer is a popular, complex test-bed for MARL , that requires most of the skills enumerated above [77, 114, 115, 131, 142]. For instance, intercepting the ball and leading it into the goal involve object retrieval and transportation skills, while the tactical placement of the players in the field is an advanced version of the coverage task.

6.3 Automated trading

Software trading agents exchange goods on electronic markets on behalf of a company or a person, using mechanisms such as negotiations and auctions. For instance, the Trading Agent Competition is a simulated contest where the agents need to arrange travel packages by bidding for goods such as plane tickets and hotel bookings [140]. Multi-agent trading can also be applied to modeling electricity markets [117].

MARL approaches to automated trading typically involve *temporal-difference* [118] or *Q-learning* agents, using approximate representations of the Q-functions to handle the large state space [48,68,125]. In some cases, cooperative agents represent the interest of a single company or individual, and merely fulfill different functions in the trading process, such as buying and selling [68]. In other cases, self-interested agents interact in parallel with the market [48, 98, 125].

6.4 Resource management

In resource management, the agents form a cooperative team, and they can be one of the following:

- Managers of resources, as in [33]. Each agent manages one resource, and the agents learn how to best service requests in order to optimize a given performance measure.
- Clients of resources, as in [102]. The agents learn how to best select resources such that a given performance measure is optimized.

A popular resource management domain is network routing [18, 28, 126]. Other examples include elevator scheduling [33] and load balancing [102]. Performance measures include average job processing times, minimum waiting time for resources, resource usage, and fairness in servicing clients.

6.5 Remarks

Though not an application domain *per se*, game-theoretic, stateless tasks are often used to test MARL approaches. Not only algorithms specifically designed for static games are tested on such tasks (e.g., *AWESOME* [31], *MetaStrategy* [93], *GIGA-WoLF* [14]), but also others that can, in principle, handle dynamic stochastic games (e.g., *EXORL* [116]).

As an avenue for future work, note that distributed control is poorly represented as a MARL application domain. This includes systems such as traffic, power, or sensor networks.

7 Example: coordinated multi-agent object transportation

In this section, several MARL algorithms are applied to an illustrative example.[5] The example, represented in Figure 6, is an abstraction of a task involving the coordinated transportation of an object by two agents. These agents (represented by numbered disks) travel on a two-dimensional discrete grid with 7×6 cells. The agents have to transport a target object (represented by a small rectangle) to the home base (delimited by a dashed black line) in minimum time, while avoiding obstacles (shown by gray blocks).

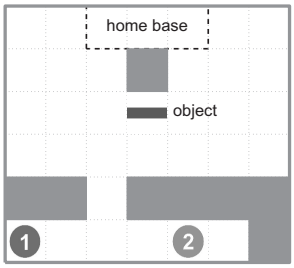


Fig. 6 The object transportation problem.

[5] A MARL software package written by the authors in MATLAB was used for this example. This package can be downloaded at http://www.dcsc.tudelft.nl/˜lbusoniu, and includes the coordinated object transportation example.

The agents start from the positions in which they are shown in Figure 6, and can move at each time step by one cell to the left, right, up, or down; they can also stand still. If the target cell is not empty, the agent does not move; similarly, if both agents attempt to move to the same cell, neither agent moves. In order to transport the target, the agents first have to grasp it. When an agent reaches a cell immediately to the left or right of the target, it automatically grasps the target; once the target is grasped, it cannot be released. Only when the two agents have grasped either side the target, they can move it. The target only moves when both agents coordinately pull in the same direction. As soon as the target has reached the home base, the trial terminates, and the agents and target are reinitialized for a new trial.

The state variables describing each agent i are its two position coordinates, $p_{i,\mathrm{X}} \in \{1,2,\ldots,7\}$, $p_{i,\mathrm{Y}} \in \{1,2,\ldots,6\}$, and a variable indicating if the agent is currently grasping the target, and if yes, to which side: $g_i \in \{\text{FREE}, \text{GRASPING-LEFT}, \text{GRASPING-RIGHT}\}$. Therefore, the complete state vector is $x = [p_{1,\mathrm{X}}, p_{1,\mathrm{Y}}, g_1, p_{2,\mathrm{X}}, p_{2,\mathrm{Y}}, g_2]^{\mathrm{T}}$. The grasping variables are needed to ensure that the state vector has the Markov property (see Section 2). Each agent's action u_i belongs to the set $U_i = \{\text{LEFT}, \text{RIGHT}, \text{UP}, \text{DOWN}, \text{STAND-STILL}\}$. So, the state space contains $|X| = (7 \cdot 6 \cdot 3)^2 = 15876$ elements, and the joint action space contains $5^2 = 25$ elements. Not all the states are valid, e.g., collisions prevent certain combinations from occurring.

The task is fully cooperative, so both agents have the same reward function, which expresses the goal of grasping the target and transporting it to the home base:

$$r_{k+1} = \begin{cases} 1 & \text{if an agent has just grasped the target} \\ 10 & \text{if the target has reached the home base} \\ 0 & \text{in all other conditions} \end{cases}$$

The discount factor is $\gamma = 0.98$. The minimum-time character of the solution results from discounting: it is better to receive the positive rewards as early as possible, otherwise discounting will decrease their contribution to the return.

The agents face two coordination problems. The first problem is to decide which of them passes first through the narrow lower passage. The second problem is to decide whether they transport the target around the left or right side of the obstacle just below the home base.

We apply three algorithms to this problem: (i) single-agent *Q-learning*, which is a representative single-agent RL algorithm, (ii) *team Q-learning*, a representative MARL algorithm for fully cooperative tasks, and (iii) *WoLF-PHC*, a representative MARL algorithm for mixed tasks. An algorithm for competitive tasks (such as *minimax-Q*) is unlikely to perform well, because the object transportation problem is fully cooperative. In any given experiment, both agents use the same algorithm, so they are homogeneous. For all the algorithms, a constant learning rate $\alpha = 0.1$ is employed, together with an ε-greedy exploration procedure. The exploration probability ε is initialized at 0.8, is constant within a trial, and decays exponentially with a factor 0.9 after every trial. For *WoLF-PHC*, the policy step sizes are $\delta_{\text{win}} = 0.1$ and $\delta_{\text{lose}} = 0.4$. Note that the Q-tables of *Q-learning* or *WoLF-PHC* agents contain $|X| \times 5 = 79380$ elements, because they only depend on that single agent's action,

whereas the Q-tables of *team Q-learning* agents depend on the joint action and therefore contain $|X| \times 5^2 = 396900$ elements.

Figure 7 shows the mean learning performance of the three algorithms across 100 independent runs, together with 95% confidence intervals on this mean. Each graph shows the evolution of the number of steps taken to reach the home base, as the number of learning trials grows. The performance is measured while the agents are learning, and the effects of exploration are included. All the algorithms quickly converge to a good performance, usually after 20 to 30 trials. Remarkably, *Q-learning* performs very well, even though a Q-learner is unaware of the other agent except through its state variables. While *team Q-learning* and *WoLF-PHC* agents do take each other into account, they do not use explicit coordination. Instead, all three algorithms achieve an *implicit* form of coordination: the agents learn to prefer one of the equally good solutions by chance, and then ignore the other solutions. The fact that explicit coordination is not required can be verified e.g., by repeating the *team Q-learning* experiment after adding social conventions. Indeed, such an algorithm produces nearly identical results to those in Figure 7. Single-agent *Q-learning* is preferable in this problem, because it provides the same performance as the other two algorithms, but requires smaller Q-tables than *team Q-learning* and has simpler update formulas than *WoLF-PHC*.

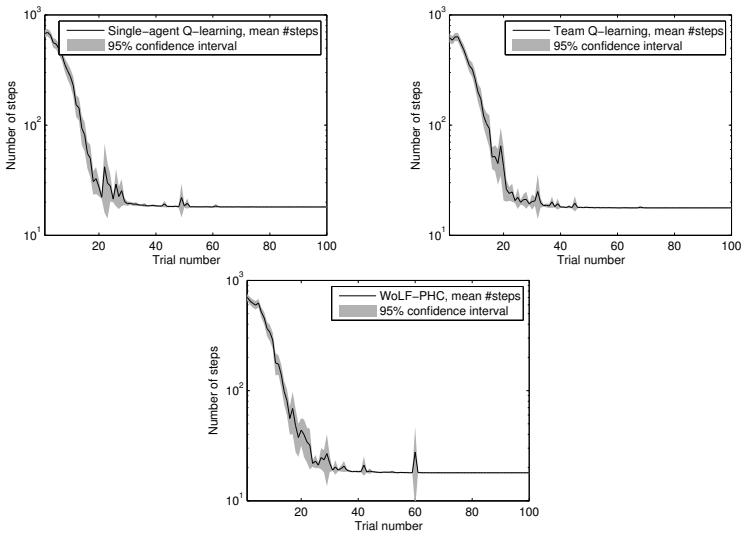


Fig. 7 Learning performance of single-agent *Q-learning*, *team Q-learning*, and *WoLF-PHC*. Note the logarithmic scale of the vertical axes.

Figure 8 shows a trajectory taken by the agents, after a representative run with *team Q-learning* (the trajectories followed by *Q-learning* and *WoLF-PHC* agents are similar). Agent 2 waits for agent 1 to go through the passage. Next, agent 1 grasps the target waiting for agent 2, and after agent 2 also arrives, they transport the target to the home base around the right of the obstacle.

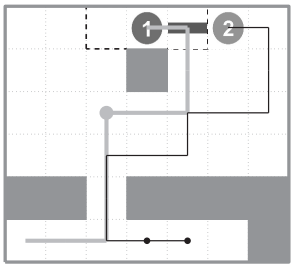


Fig. 8 A solution obtained with *team Q-learning*. Agent 1 travels along the thick gray line, and agent 2 along the thin black line. Dots mark the states where the agents stand still; the size of each dot is proportional with the time spent standing still in that cell.

8 Outlook

In this section, we discuss some general open issues in MARL , concerning the suitability of MARL algorithms in practice, the choice of the multi-agent learning goal, and the study of the joint environment and learning dynamics.

8.1 Practical MARL

MARL algorithms are typically applied to small problems only, like static games and small grid-worlds (like the grid-world of Section 7). As a consequence, it is unclear whether these algorithms scale up to realistic multi-agent problems, where the state and action spaces are large or even continuous. Few algorithms are able to deal with incomplete, uncertain observations. This situation can be explained by noting that scalability and dealing with imperfect observations are also open problems in single-agent RL . Nevertheless, improving the suitability of MARL to problems of practical interest is an essential research step. Below, we describe several directions in which this research can proceed, and point to some pioneering work done along these directions.

Scalability is a central concern for MARL . Most algorithms require explicit tabular storage of the agents' Q-functions and possibly of their policies. This means they only work in problems with a relatively small number of discrete states and actions. When the state and action spaces contain a large or infinite number of elements (e.g., when they are continuous), the tabular storage of the Q-function becomes impractical or impossible. Instead, the Q-function must be represented approximately. Approximate MARL algorithms have been proposed e.g., for discrete, large state-action spaces [1], for continuous states and discrete actions [22, 52], and for continuous states and actions [35, 122]. Unfortunately, many of these algorithms only work in restricted classes of problems and are heuristic in nature. Significant advances in approximate MARL can be made by putting to use the extensive results on single-agent approximate RL , which are outlined next.

While offline, batch algorithms have been very successful in single-agent approximate RL [34, 66, 80, 83, 85, 99], they are impractical in MARL , because an offline solution can become inappropriate as the behavior of the other agents changes. Instead, online approximate RL algorithms are required, such as approximate variants of *Q-learning*. *Q-learning* has been combined with Q-function approximators relying on, e.g., basis-function representations [110, 121], neural networks and self-organizing maps [127], and fuzzy rule bases [41, 47, 56]. Approximate *Q-learning* is provably convergent if a linearly parameterized approximator is employed, and under the restrictive condition that the agent's policy is kept fixed while learning [76]. Linearly parameterized approximators include basis-function representations, as well as certain classes of neural networks and fuzzy rule bases. An alternative to using *Q-learning* is to develop online, incremental versions of the successful offline, batch algorithms, see e.g., [57]. Another promising category of algorithms consists of *actor-critic* techniques, many of which are specialized for continuous state and action spaces [7, 10, 63, 90]. The reader interested in approximate RL is referred to [9, 23, 92], to Chapter 8 of [120], and to Chapter 6 of [8].

A complementary avenue for improving scalability is the discovery and exploitation of decentralized, modular structure in the multi-agent task [21, 38, 43].

Incomplete, uncertain state measurements could be handled with techniques related to partially observable Markov decision processes [72], as in [44, 51].

Providing *domain knowledge* to the agents can greatly help them to learn solutions of realistic tasks. In contrast, the large size of the state-action space and the delays in receiving informative rewards mean that MARL without any prior knowledge is very slow. Domain knowledge can be supplied in several forms. Informative reward functions, also rewarding promising behaviors rather than just the achievement of the goal, can be provided to the agents [74,75]. Humans or skilled agents can teach unskilled agents how to solve the task [94]. Shaping is a technique whereby the learning process starts by presenting the agents with simpler tasks, and progressively moves toward complex ones [24]. Preprogrammed reflex behaviors can be built into the agents [74, 75]. Knowledge about the task structure can be used to decompose it in subtasks, and learn a modular solution with e.g., hierarchical RL [40]. If approximate solutions are used, a good way to incorporate domain knowledge is to structure the approximator in a way that ensures high accuracy in important regions of the state-action space, e.g., close to the goal. Last, but not least, if a (possibly incomplete) task model is available, this model can be used with model-based RL algorithms to initialize Q-functions to reasonable, rather than arbitrary, values.

8.2 Learning goal

Defining a suitable MARL goal for general, dynamic stochastic games is a difficult open problem. MARL goals are typically formulated in terms of static games. Their extension to dynamic tasks, as discussed in Section 4, is not always clear or even possible.

Stability of the learning process is needed, because the behavior of stable agents is more amenable to analysis and meaningful performance guarantees. Adaptation to the other agents is needed because their behavior is generally unpredictable. Therefore, a good multi-agent learning goal must include both components. This means that MARL algorithms should not be totally independent of the other agents, nor just track their behavior without concern for convergence.

Moreover, from a practical viewpoint, a realistic learning goal should include bounds on the transient performance, in addition to the usual asymptotic requirements. Examples of such bounds include maximum time constraints for reaching a desired performance level, or a lower bound on the instantaneous performance. Some steps in this direction have been taken in [14, 93].

8.3 Joint environment and learning dynamics

So far, game-theory-based analysis has only been applied to the learning dynamics of the agents [130, 132, 134], while the dynamics of the environment have not been explicitly considered. Tools developed in the area of robust control can play an important role in the analysis of the learning process as a whole, comprising the interacting environment and learning dynamics. In addition, this framework can incorporate prior knowledge about bounds on imperfect observations, such as noise-corrupted variables; and can help study the robustness of MARL algorithms against uncertainty in the other agents' dynamics.

9 Related work

Besides its heritage relationship with single-agent RL , the MARL field has strong connections with game theory, evolutionary computation, and more generally with the direct optimization of agent policies. These relationships are described next.

Game theory [3] and especially the theory of learning in games [39] make an essential contribution to MARL . In this chapter, we have reviewed relevant game-theoretic algorithms for static and repeated games, and we have investigated the contribution of game theory to MARL algorithms for dynamic tasks. Other authors have investigated more closely the relationship between game theory and MARL . For instance, Bowling and Veloso [17] have discussed several MARL algorithms, showing that these algorithms combine temporal-difference RL with game-theoretic solvers for the static games that arise in each state of the dynamic stochastic game. Shoham et al. [108] have critically examined the focus of MARL research on game-theoretic equilibria, using a selection of MARL algorithms to illustrate their arguments.

Rather than estimating value functions and using them to derive policies, it is also possible to directly explore the space of agent behaviors using, e.g., nonlinear

optimization techniques. Evolutionary multi-agent learning is a prominent example of such an approach. Evolutionary computation applies principles of biological evolution to the search for solutions (agent behaviors) of the given task [2, 55]. Populations of candidate behaviors are maintained. Candidates are evaluated using a fitness function related to the return, and selected for breeding or mutation on the basis of their fitness. Panait and Luke [86] have offered a comprehensive survey of evolutionary learning and MARL for cooperative agent teams. For the interested reader, examples of co-evolution techniques, where the behaviors of the agents evolve in parallel, can be found in [36, 87, 91]. Complementary, team learning techniques, where the entire set of agent behaviors is discovered by a single evolution process, can be found e.g., in [45, 78, 101]. Besides evolutionary computation, other approaches to the direct optimization of agent behaviors are gradient search [65], probabilistic hill-climbing [46], as well as more general behavior modification heuristics [103]. Because direct behavior optimization cannot readily benefit from the structure of the RL task, we did not focus on it in this chapter. Instead, we have mainly discussed the contribution of direct behavior optimization to MARL algorithms based on temporal-difference RL .

Evolutionary game theory sits at the intersection of evolutionary learning and game theory [111]. Tuyls and Nowé [132] have investigated the relationship between MARL and evolutionary game theory, focusing on static tasks.

10 Conclusions

Multi-agent reinforcement learning (MARL) is a young, but active and rapidly expanding field of research. MARL aims to provide an array of algorithms that enable multiple agents to learn the solution of difficult tasks, using limited or no prior knowledge. To this end, MARL integrates results from single-agent RL , game theory, and direct policy search.

This chapter has provided an extensive overview of MARL . First, we have presented the main benefits and challenges of MARL , as well as the different viewpoints on defining the MARL learning goal. Then, we have discussed in detail a representative set of MARL algorithms for fully cooperative, fully competitive, and mixed tasks. We have focused on autonomous agents that learn how to solve dynamic tasks online, with algorithms originating in temporal-difference RL , but we have also investigated techniques for static tasks. Additionally, we have reviewed some representative problem domains where MARL techniques have been applied, and we have illustrated the behavior of several MARL algorithms in a simulation example involving multi-agent object transportation.

Many avenues for MARL are open at this point, and many research opportunities present themselves. We have provided an outlook synthesizing these open issues and opportunities. In particular, approximate RL is needed to apply MARL to realistic problems, and control theory can help analyzing the learning dynamics and assessing the robustness to uncertainty in the observations or in the other agents'

behavior. In our view, significant progress in the field of multi-agent learning can be achieved by a more intensive cross-fertilization between the fields of machine learning, game theory, and control theory.

Acknowledgements This work was financially supported by the BSIK project "Interactive Collaborative Information Systems" (grant no. BSIK03024).

References

1. Abul, O., Polat, F., Alhajj, R.: Multiagent reinforcement learning using function approximation. IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews **4**(4), 485–497 (2000)
2. Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press (1996)
3. Başar, T., Olsder, G.J.: Dynamic Noncooperative Game Theory, 2nd edn. Society for Industrial and Applied Mathematics (SIAM) (1999)
4. Bakker, B., Steingrover, M., Schouten, R., Nijhuis, E., Kester, L.: Cooperative multi-agent reinforcement learning of traffic lights. In: Workshop on Cooperative Multi-Agent Learning, 16th European Conference on Machine Learning (ECML-05). Porto, Portugal (2005)
5. Banerjee, B., Peng, J.: Adaptive policy gradient in multiagent learning. In: Proceedings 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03), pp. 686–692. Melbourne, Australia (2003)
6. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics **13**(5), 833–846 (1983)
7. Berenji, H.R., Vengerov, D.: A convergent actor-critic-based FRL algorithm with application to power management of wireless transmitters. IEEE Transactions on Fuzzy Systems **11**(4), 478–485 (2003)
8. Bertsekas, D.P.: Dynamic Programming and Optimal Control, vol. 2, 3rd edn. Athena Scientific (2007)
9. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific (1996)
10. Borkar, V.: An actor-critic algorithm for constrained Markov decision processes. Systems & Control Letters **54**(3), 207–213 (2005)
11. Boutilier, C.: Planning, learning and coordination in multiagent decision processes. In: Proceedings 6th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-96), pp. 195–210. De Zeeuwse Stromen, The Netherlands (1996)
12. Bowling, M.: Convergence problems of general-sum multiagent reinforcement learning. In: Proceedings 17th International Conference on Machine Learning (ICML-00), pp. 89–94. Stanford University, US (2000)
13. Bowling, M.: Multiagent learning in the presence of agents with limitations. Ph.D. thesis, Computer Science Dept., Carnegie Mellon University, Pittsburgh, US (2003)
14. Bowling, M.: Convergence and no-regret in multiagent learning. In: L.K. Saul, Y. Weiss, L. Bottou (eds.) Advances in Neural Information Processing Systems 17, pp. 209–216. MIT Press (2005)
15. Bowling, M., Veloso, M.: An analysis of stochastic game theory for multiagent reinforcement learning. Tech. rep., Computer Science Dept., Carnegie Mellon University, Pittsburgh, US (2000). URL http://www.cs.ualberta.ca/ bowling/papers/00tr.pdf
16. Bowling, M., Veloso, M.: Rational and convergent learning in stochastic games. In: Proceedings 17th International Conference on Artificial Intelligence (IJCAI-01), pp. 1021–1026. San Francisco, US (2001)

17. Bowling, M., Veloso, M.: Multiagent learning using a variable learning rate. Artificial Intelligence **136**(2), 215–250 (2002)
18. Boyan, J.A., Littman, M.L.: Packet routing in dynamically changing networks: A reinforcement learning approach. In: J. Moody (ed.) Advances in Neural Information Processing Systems 6, pp. 671–678. Morgan Kaufmann (1994)
19. Brown, G.W.: Iterative solutions of games by fictitious play. In: T.C. Koopmans (ed.) Activitiy Analysis of Production and Allocation, chap. XXIV, pp. 374–376. Wiley (1951)
20. Buşoniu, L., Babuška, R., De Schutter, B.: A comprehensive survey of multi-agent reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews **38**(2), 156–172 (2008)
21. Buşoniu, L., De Schutter, B., Babuška, R.: Multiagent reinforcement learning with adaptive state focus. In: Proceedings 17th Belgian-Dutch Conference on Artificial Intelligence (BNAIC-05), pp. 35–42. Brussels, Belgium (2005)
22. Buşoniu, L., De Schutter, B., Babuška, R.: Decentralized reinforcement learning control of a robotic manipulator. In: Proceedings 9th International Conference of Control, Automation, Robotics, and Vision (ICARCV-06), pp. 1347–1352. Singapore (2006)
23. Buşoniu, L., De Schutter, B., Babuška, R.: Approximate dynamic programming and reinforcement learning. In: R. Babuška, F.C.A. Groen (eds.) Interactive Collaborative Information Systems, no. 281 in Studies in Computational Intelligence. Springer (2010). To appear.
24. Buffet, O., Dutech, A., Charpillet, F.: Shaping multi-agent systems with gradient reinforcement learning. Autonomous Agents and Multi-Agent Systems **15**(2), 197–220 (2007)
25. Carmel, D., Markovitch, S.: Opponent modeling in multi-agent systems. In: G. Weiß, S. Sen (eds.) Adaptation and Learning in Multi-Agent Systems, chap. 3, pp. 40–52. Springer Verlag (1996)
26. Chalkiadakis, G.: Multiagent reinforcement learning: Stochastic games with multiple learning players. Tech. rep., Dept. of Computer Science, University of Toronto, Canada (2003). URL http://www.cs.toronto.edu/~gehalk/DepthReport/DepthReport.ps
27. Cherkassky, V., Mulier, F.: Learning from Data: Concepts, Theory, And Methods. Wiley (1998)
28. Choi, S.P.M., Yeung, D.Y.: Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In: Advances in Neural Information Processing Systems 8 (NIPS-95), pp. 945–951. Denver, US (1995)
29. Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. In: Proceedings 15th National Conference on Artificial Intelligence and 10th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-98), pp. 746–752. Madison, US (1998)
30. Clouse, J.: Learning from an automated training agent. In: Working Notes Workshop on Agents that Learn from Other Agents, 12th International Conference on Machine Learning (ICML-95). Tahoe City, US (1995)
31. Conitzer, V., Sandholm, T.: AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In: Proceedings 20th International Conference on Machine Learning (ICML-03), pp. 83–90. Washington, US (2003)
32. Crites, R.H., Barto, A.G.: Improving elevator performance using reinforcement learning. In: D.S. Touretzky, M.C. Mozer, M.E. Hasselmo (eds.) Advances in Neural Information Processing Systems 8, pp. 1017–1023. MIT Press (1996)
33. Crites, R.H., Barto, A.G.: Elevator group control using multiple reinforcement learning agents. Machine Learning **33**(2–3), 235–262 (1998)
34. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research **6**, 503–556 (2005)
35. Fernández, F., Parker, L.E.: Learning in large cooperative multi-robot systems. International Journal of Robotics and Automation **16**(4), 217–226 (2001). Special Issue on Computational Intelligence Techniques in Cooperative Robots

36. Ficici, S.G., Pollack, J.B.: A game-theoretic approach to the simple coevolutionary algorithm. In: Proceedings 6th International Conference Parallel Problem Solving from Nature (PPSN-VI), pp. 467–476. Paris, France (2000)
37. Fischer, F., Rovatsos, M., Weiss, G.: Hierarchical reinforcement learning in communication-mediated multiagent coordination. In: Proceedings 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-04), pp. 1334–1335. New York, US (2004)
38. Fitch, R., Hengst, B., Suc, D., Calbert, G., Scholz, J.B.: Structural abstraction experiments in reinforcement learning. In: Proceedings 18th Australian Joint Conference on Artificial Intelligence (AI-05), *Lecture Notes in Computer Science*, vol. 3809, pp. 164–175. Sydney, Australia (2005)
39. Fudenberg, D., Levine, D.K.: The Theory of Learning in Games. MIT Press (1998)
40. Ghavamzadeh, M., Mahadevan, S., Makar, R.: Hierarchical multi-agent reinforcement learning. Autonomous Agents and Multi-Agent Systems **13**(2), 197–229 (2006)
41. Glorennec, P.Y.: Reinforcement learning: An overview. In: Proceedings European Symposium on Intelligent Techniques (ESIT-00), pp. 17–35. Aachen, Germany (2000)
42. Greenwald, A., Hall, K.: Correlated-Q learning. In: Proceedings 20th International Conference on Machine Learning (ICML-03), pp. 242–249. Washington, US (2003)
43. Guestrin, C., Lagoudakis, M.G., Parr, R.: Coordinated reinforcement learning. In: Proceedings 19th International Conference on Machine Learning (ICML-02), pp. 227–234. Sydney, Australia (2002)
44. Hansen, E.A., Bernstein, D.S., Zilberstein, S.: Dynamic programming for partially observable stochastic games. In: Proceedings 19th National Conference on Artificial Intelligence (AAAI-04), pp. 709–715. San Jose, US (2004)
45. Haynes, T., Wainwright, R., Sen, S., Schoenefeld, D.: Strongly typed genetic programming in evolving cooperation strategies. In: Proceedings 6th International Conference on Genetic Algorithms (ICGA-95), pp. 271–278. Pittsburgh, US (1995)
46. Ho, F., Kamel, M.: Learning coordination strategies for cooperative multiagent systems. Machine Learning **33**(2–3), 155–177 (1998)
47. Horiuchi, T., Fujino, A., Katai, O., Sawaragi, T.: Fuzzy interpolation-based Q-learning with continuous states and actions. In: Proceedings 5th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE-96), pp. 594–600. New Orleans, US (1996)
48. Hsu, W.T., Soo, V.W.: Market performance of adaptive trading agents in synchronous double auctions. In: Proceedings 4th Pacific Rim International Workshop on Multi-Agents. Intelligent Agents: Specification, Modeling, and Applications (PRIMA-01), *Lecture Notes in Computer Science*, vol. 2132, pp. 108–121. Taipei, Taiwan (2001)
49. Hu, J., Wellman, M.P.: Multiagent reinforcement learning: Theoretical framework and an algorithm. In: Proceedings 15th International Conference on Machine Learning (ICML-98), pp. 242–250. Madison, US (1998)
50. Hu, J., Wellman, M.P.: Nash Q-learning for general-sum stochastic games. Journal of Machine Learning Research **4**, 1039–1069 (2003)
51. Ishii, S., Fujita, H., Mitsutake, M., Yamazaki, T., Matsuda, J., Matsuno, Y.: A reinforcement learning scheme for a partially-observable multi-agent game. Machine Learning **59**(1–2), 31–54 (2005)
52. Ishiwaka, Y., Sato, T., Kakazu, Y.: An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning. Robotics and Autonomous Systems **43**(4), 245–256 (2003)
53. Jaakkola, T., Jordan, M.I., Singh, S.P.: On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation **6**(6), 1185–1201 (1994)
54. Jafari, A., Greenwald, A.R., Gondek, D., Ercal, G.: On no-regret learning, fictitious play, and Nash equilibrium. In: Proceedings 18th International Conference on Machine Learning (ICML-01), pp. 226–233. Williams College, Williamstown, US (2001)
55. Jong, K.D.: Evolutionary Computation: A Unified Approach. MIT Press (2005)
56. Jouffe, L.: Fuzzy inference system learning by reinforcement methods. IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews **28**(3), 338–355 (1998)

57. Jung, T., Polani, D.: Kernelizing LSPE(λ). In: Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-07), pp. 338–345. Honolulu, US (2007)
58. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research **4**, 237–285 (1996)
59. Kapetanakis, S., Kudenko, D.: Reinforcement learning of coordination in cooperative multi-agent systems. In: Proceedings 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI-02), pp. 326–331. Menlo Park, US (2002)
60. Kok, J.R., 't Hoen, P.J., Bakker, B., Vlassis, N.: Utile coordination: Learning interdependencies among cooperative agents. In: Proceedings IEEE Symposium on Computational Intelligence and Games (CIG-05), pp. 29–36. Colchester, United Kingdom (2005)
61. Kok, J.R., Spaan, M.T.J., Vlassis, N.: Non-communicative multi-robot coordination in dynamic environment. Robotics and Autonomous Systems **50**(2–3), 99–114 (2005)
62. Kok, J.R., Vlassis, N.: Sparse cooperative Q-learning. In: Proceedings 21st International Conference on Machine Learning (ICML-04), pp. 481–488. Banff, Canada (2004)
63. Konda, V.R., Tsitsiklis, J.N.: On actor-critic algorithms. SIAM Journal on Control and Optimization **42**(4), 1143–1166 (2003)
64. Könönen, V.: Asymmetric multiagent reinforcement learning. In: Proceedings IEEE/WIC International Conference on Intelligent Agent Technology (IAT-03), pp. 336–342. Halifax, Canada (2003)
65. Könönen, V.: Gradient based method for symmetric and asymmetric multiagent reinforcement learning. In: Proceedings 4th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL-03), pp. 68–75. Hong Kong, China (2003)
66. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. Journal of Machine Learning Research **4**, 1107–1149 (2003)
67. Lauer, M., Riedmiller, M.: An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In: Proceedings 17th International Conference on Machine Learning (ICML-00), pp. 535–542. Stanford University, US (2000)
68. Lee, J.W., O, J.: A multi-agent Q-learning framework for optimizing stock trading systems. In: 13th International Conference Database and Expert Systems Applications (DEXA-02), *Lecture Notes in Computer Science*, vol. 2453, pp. 153–162. Aix-en-Provence, France (2002)
69. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: Proceedings 11th International Conference on Machine Learning (ICML-94), pp. 157–163. New Brunswick, US (1994)
70. Littman, M.L.: Value-function reinforcement learning in Markov games. Journal of Cognitive Systems Research **2**(1), 55–66 (2001)
71. Littman, M.L., Stone, P.: Implicit negotiation in repeated games. In: Proceedings 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), pp. 96–105. Seattle, US (2001)
72. Lovejoy, W.S.: Computationally feasible bounds for partially observed Markov decision processes. Operations Research **39**(1), 162–175 (1991)
73. Matarić, M.J.: Reward functions for accelerated learning. In: Proceedings 11th International Conference on Machine Learning (ICML-94), pp. 181–189. New Brunswick, US (1994)
74. Matarić, M.J.: Learning in multi-robot systems. In: G. Weiß, S. Sen (eds.) Adaptation and Learning in Multi–Agent Systems, chap. 10, pp. 152–163. Springer Verlag (1996)
75. Matarić, M.J.: Reinforcement learning in the multi-robot domain. Autonomous Robots **4**(1), 73–83 (1997)
76. Melo, F.S., Meyn, S.P., Ribeiro, M.I.: An analysis of reinforcement learning with function approximation. In: Proceedings 25th International Conference on Machine Learning (ICML-08), pp. 664–671. Helsinki, Finland. (2008)
77. Merke, A., Riedmiller, M.A.: Karlsruhe brainstormers – A reinforcement learning approach to robotic soccer. In: Robot Soccer World Cup V (RoboCup 2001), *Lecture Notes in Computer Science*, vol. 2377, pp. 435–440. Washington, US (2001)

78. Miconi, T.: When evolving populations is better than coevolving individuals: The blind mice problem. In: Proceedings 18th International Joint Conference on Artificial Intelligence (IJCAI-03), pp. 647–652. Acapulco, Mexico (2003)
79. Moore, A.W., Atkeson, C.G.: Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning **13**, 103–130 (1993)
80. Munos, R., Szepesvári, Cs.: Finite time bounds for fitted value iteration. Journal of Machine Learning Research **9**, 815–857 (2008)
81. Nagendra Prasad, M.V., Lesser, V.R., Lander, S.E.: Learning organizational oles for negotiated search in a multiagent system. International Journal of Human-Computer Studies **48**(1), 51–67 (1998)
82. Nash, S., Sofer, A.: Linear and Nonlinear Programming. McGraw-Hill (1996)
83. Nedić, A., Bertsekas, D.P.: Least-squares policy evaluation algorithms with linear function approximation. Discrete Event Dynamic Systems: Theory and Applications **13**(1–2), 79–110 (2003)
84. Negenborn, R.R., De Schutter, B., Hellendoorn, H.: Multi-agent model predictive control for transportation networks: Serial versus parallel schemes. Engineering Applications of Artificial Intelligence **21**(3), 353–366 (2008)
85. Ormoneit, D., Sen, S.: Kernel-based reinforcement learning. Machine Learning **49**(2–3), 161–178 (2002)
86. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Autonomous Agents and Multi-Agent Systems **11**(3), 387–434 (2005)
87. Panait, L., Wiegand, R.P., Luke, S.: Improving coevolutionary search for optimal multiagent behaviors. In: Proceedings 18th International Joint Conference on Artificial Intelligence (IJCAI-03), pp. 653–660. Acapulco, Mexico (2003)
88. Parunak, H.V.D.: Industrial and practical applications of DAI. In: G. Weiss (ed.) Multi–Agent Systems: A Modern Approach to Distributed Artificial Intelligence, chap. 9, pp. 377–412. MIT Press (1999)
89. Peng, J., Williams, R.J.: Incremental multi-step Q-learning. Machine Learning **22**(1–3), 283–290 (1996)
90. Peters, J., Schaal, S.: Natural actor-critic. Neurocomputing **71**(7–9), 1180–1190 (2008)
91. Potter, M.A., Jong, K.A.D.: A cooperative coevolutionary approach to function optimization. In: Proceedings 3rd Conference on Parallel Problem Solving from Nature (PPSN-III), pp. 249–257. Jerusalem, Israel (1994)
92. Powell, W.B.: Approximate Dynamic Programming: Solving the Curses of Dimensionality. Wiley (2007)
93. Powers, R., Shoham, Y.: New criteria and a new algorithm for learning in multi-agent systems. In: L.K. Saul, Y. Weiss, L. Bottou (eds.) Advances in Neural Information Processing Systems 17, pp. 1089–1096. MIT Press (2005)
94. Price, B., Boutilier, C.: Implicit imitation in multiagent reinforcement learning. In: Proceedings 16th International Conference on Machine Learning (ICML-99), pp. 325–334. Bled, Slovenia (1999)
95. Price, B., Boutilier, C.: Accelerating reinforcement learning through implicit imitation. Journal of Artificial Intelligence Research **19**, 569–629 (2003)
96. Puterman, M.L.: Markov Decision Processes—Discrete Stochastic Dynamic Programming. Wiley (1994)
97. Pynadath, D.V., Tambe, M.: The communicative multiagent team decision problem: Analyzing teamwork theories and models. Journal of Artificial Intelligence Research **16**, 389–423 (2002)
98. Raju, C., Narahari, Y., Ravikumar, K.: Reinforcement learning applications in dynamic pricing of retail markets. In: Proceedings 2003 IEEE International Conference on E-Commerce (CEC-03), pp. 339–346. Newport Beach, US (2003)
99. Riedmiller, M.: Neural fitted Q-iteration – first experiences with a data efficient neural reinforcement learning method. In: Proceedings 16th European Conference on Machine Learning (ECML-05), *Lecture Notes in Computer Science*, vol. 3720, pp. 317–328. Porto, Portugal (2005)

100. Riedmiller, M.A., Moore, A.W., Schneider, J.G.: Reinforcement learning for cooperating and communicating reactive agents in electrical power grids. In: M. Hannebauer, J. Wendler, E. Pagello (eds.) Balancing Reactivity and Social Deliberation in Multi-Agent Systems, pp. 137–149. Springer (2000)
101. Salustowicz, R., Wiering, M., Schmidhuber, J.: Learning team strategies: Soccer case studies. Machine Learning **33**(2–3), 263–282 (1998)
102. Schaerf, A., Shoham, Y., Tennenholtz, M.: Adaptive load balancing: A study in multi-agent learning. Journal of Artificial Intelligence Research **2**, 475–500 (1995)
103. Schmidhuber, J.: A general method for incremental self-improvement and multi-agent learning. In: X. Yao (ed.) Evolutionary Computation: Theory and Applications, chap. 3, pp. 81–123. World Scientific (1999)
104. Sejnowski, T.J., Hinton, G.E. (eds.): Unsupervised Learning: Foundations of Neural Computation. MIT Press (1999)
105. Sen, S., Sekaran, M., Hale, J.: Learning to coordinate without sharing information. In: Proceedings 12th National Conference on Artificial Intelligence (AAAI-94), pp. 426–431. Seattle, US (1994)
106. Sen, S., Weiss, G.: Learning in multiagent systems. In: G. Weiss (ed.) Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, chap. 6, pp. 259–298. MIT Press (1999)
107. Shoham, Y., Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations. Cambridge University Press (2008)
108. Shoham, Y., Powers, R., Grenager, T.: If multi-agent learning is the answer, what is the question? Artificial Intelligence **171**(7), 365–377 (2007)
109. Singh, S., Kearns, M., Mansour, Y.: Nash convergence of gradient dynamics in general-sum games. In: Proceedings 16th Conference on Uncertainty in Artificial Intelligence (UAI-00), pp. 541–548. San Francisco, US (2000)
110. Singh, S.P., Jaakkola, T., Jordan, M.I.: Reinforcement learning with soft state aggregation. In: G. Tesauro, D.S. Touretzky, T.K. Leen (eds.) Advances in Neural Information Processing Systems 7, pp. 361–368. MIT Press (1995)
111. Smith, J.M.: Evolution and the Theory of Games. Cambridge University Press (1982)
112. Spaan, M.T.J., Vlassis, N., Groen, F.C.A.: High level coordination of agents based on multiagent Markov decision processes with roles. In: Workshop on Cooperative Robotics, 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-02), pp. 66–73. Lausanne, Switzerland (2002)
113. Stephan, V., Debes, K., Gross, H.M., Wintrich, F., Wintrich, H.: A reinforcement learning based neural multi-agent-system for control of a combustion process. In: Proceedings IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN-00), pp. 6217–6222. Como, Italy (2000)
114. Stone, P., Veloso, M.: Team-partitioned, opaque-transition reinforcement learning. In: Proceedings 3rd International Conference on Autonomous Agents (Agents-99), pp. 206–212. Seattle, US (1999)
115. Stone, P., Veloso, M.: Multiagent systems: A survey from the machine learning perspective. Autonomous Robots **8**(3), 345–383 (2000)
116. Suematsu, N., Hayashi, A.: A multiagent reinforcement learning algorithm using extended optimal response. In: Proceedings 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-02), pp. 370–377. Bologna, Italy (2002)
117. Sueyoshi, T., Tadiparthi, G.R.: An agent-based decision support system for wholesale electricity markets. Decision Support Systems **44**, 425–446 (2008)
118. Sutton, R.S.: Learning to predict by the method of temporal differences. Machine Learning **3**, 9–44 (1988)
119. Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings 7th International Conference on Machine Learning (ICML-90), pp. 216–224. Austin, US (1990)
120. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)

121. Szepesvári, Cs., Smart, W.D.: Interpolation-based Q-learning. In: Proceedings 21st International Conference on Machine Learning (ICML-04), pp. 791–798. Bannf, Canada (2004)
122. Tamakoshi, H., Ishii, S.: Multiagent reinforcement learning applied to a chase problem in a continuous world. Artificial Life and Robotics **5**(4), 202–206 (2001)
123. Tan, M.: Multi-agent reinforcement learning: Independent vs. cooperative agents. In: Proceedings 10th International Conference on Machine Learning (ICML-93), pp. 330–337. Amherst, US (1993)
124. Tesauro, G.: Extending Q-learning to general adaptive multi-agent systems. In: S. Thrun, L.K. Saul, B. Schölkopf (eds.) Advances in Neural Information Processing Systems 16. MIT Press (2004)
125. Tesauro, G., Kephart, J.O.: Pricing in agent economies using multi-agent Q-learning. Autonomous Agents and Multi-Agent Systems **5**(3), 289–304 (2002)
126. Tillotson, P., Wu, Q., Hughes, P.: Multi-agent learning for routing control within an Internet environment. Engineering Applications of Artificial Intelligence **17**(2), 179–185 (2004)
127. Touzet, C.F.: Neural reinforcement learning for behaviour synthesis. Robotics and Autonomous Systems **22**(3–4), 251–81 (1997)
128. Touzet, C.F.: Robot awareness in cooperative mobile robot learning. Autonomous Robots **8**(1), 87–97 (2000)
129. Tsitsiklis, J.N.: Asynchronous stochastic approximation and Q-learning. Machine Learning **16**(1), 185–202 (1994)
130. Tuyls, K., 't Hoen, P.J., Vanschoenwinkel, B.: An evolutionary dynamical analysis of multi-agent learning in iterated games. Autonomous Agents and Multi-Agent Systems **12**(1), 115–153 (2006)
131. Tuyls, K., Maes, S., Manderick, B.: Q-learning in simulated robotic soccer – large state spaces and incomplete information. In: Proceedings 2002 International Conference on Machine Learning and Applications (ICMLA-02), pp. 226–232. Las Vegas, US (2002)
132. Tuyls, K., Nowé, A.: Evolutionary game theory and multi-agent reinforcement learning. The Knowledge Engineering Review **20**(1), 63–90 (2005)
133. Uther, W.T., Veloso, M.: Adversarial reinforcement learning. Tech. rep., School of Computer Science, Carnegie Mellon University, Pittsburgh, US (1997). URL http://www.cs.cmu.edu/afs/cs/user/will/www/papers/Uther97a.ps
134. Vidal, J.M.: Learning in multiagent systems: An introduction from a game-theoretic perspective. In: E. Alonso (ed.) Adaptive Agents: Lecture Notes in Artificial Intelligence, vol. 2636, pp. 202–215. Springer Verlag (2003)
135. Vlassis, N.: A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. Synthesis Lectures in Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers (2007)
136. Wang, X., Sandholm, T.: Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In: S. Becker, S. Thrun, K. Obermayer (eds.) Advances in Neural Information Processing Systems 15, pp. 1571–1578. MIT Press (2003)
137. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8**, 279–292 (1992)
138. Weinberg, M., Rosenschein, J.S.: Best-response multiagent learning in non-stationary environments. In: Proceedings 3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-04), pp. 506–513. New York, US (2004)
139. Weiss, G. (ed.): Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press (1999)
140. Wellman, M.P., Greenwald, A.R., Stone, P., Wurman, P.R.: The 2001 Trading Agent Competition. Electronic Markets **13**(1) (2003)
141. Wiering, M.: Multi-agent reinforcement learning for traffic light control. In: Proceedings 17th International Conference on Machine Learning (ICML-00), pp. 1151–1158. Stanford University, US (2000)
142. Wiering, M., Salustowicz, R., Schmidhuber, J.: Reinforcement learning soccer teams with incomplete world models. Autonomous Robots **7**(1), 77–88 (1999)
143. Zapechelnyuk, A.: Limit behavior of no-regret dynamics. Discussion Papers 21, Kyiv School of Economics, Kyiv, Ucraine (2009)

144. Zinkevich, M.: Online convex programming and generalized infinitesimal gradient ascent. In: Proceedings 20th International Conference on Machine Learning (ICML-03), pp. 928–936. Washington, US (2003)