# Markov Games and Reinforcement Learning

Ismini Lourentzou

Univesity of Illinois at Urbana - Champaign
lourent2@illinois.edu

**Abstract.** We survey recent work on multi-agent reinforcement learning (MARL), which is occupied mostly with stochastic games. We briefly mention the single agent approach where the presence of other agents is considered part of the environment and present work on extending this scenario to multiple agents. These situations arise in a wide spectrum of domains such as robotics, economics, distributed control, auctions, communications etc., where the optimal policy of an agent is dependent on the policies of other agents. In such cases, and especially when the overall control of the system is decentralized, coordination with other agents or modeling the conflicting objectives is crucial. Since the field of MARL is very broad, we focus on techniques that exploit the structure of the RL problem by learning value functions as a first step towards learning more about this research area. This paper also provides MARL algorithms where Game Theory has made essential contributions and we argue that a closer connection between these two fields in necessary for advancing the field.

**Keywords:** Markov Games, Stochastic Games, Reinforcement Learning, Multi-agent Learning

## 1 Introduction

Multi-agent systems model dynamic and nondeterministic environments that solve complex problems in a variety of applications such as financial markets, traffic control, robotics, distributed systems, resource allocation, smart grids etc.

When systems are inherently decentralized, either in the case of distributed systems where data are located in different servers or when there is no central controlling component to coordinate the request in resources from subcomponents, or simply in the general case of dealing with multiple, possibly conflicting, objectives, single-agent reinforcement learning cannot fully model and exploit such dependencies. Many examples fall into this category, with most interesting ones to be robot interaction (for example in Robocup), load balancing or even web search engines [10].

Moreover, in environments where coordination of agents is needed, sharing experience can help agents to accomplish shared or similar tasks faster and better, as well as increase scalability of the system; in the case of failure of one agent, the task can be assigned to another agent or we can also easily insert new agents into the system.

In these situations we are dealing with agents that not only have to learn an optimal behavior or policy, but also have to adapt to new situations, where we assume a changing environment that requires multiple sequential decisions. In such a complex task definition, agents are required to coordinate while taking into consideration the state of their environment.

This scenario becomes increasingly complicated when we are dealing with limited or incomplete information about the overall system. More particularly, an agent may not even be aware of the presence of other agents, making the environment non-stationary. Even in the case of complete information, learning a fully joint state-action space can be computational intractable with respect to the number of agents and the level of coordination required between them.

There are many categorization schemes of Markov Games: independent learning versus joint action learning, general Markov games versus normal form games, continuous action space versus discrete action space etc. In this survey, we present recent work that extends the single-agent reinforcement learning framework by borrowing results and equilibrium definitions from Game Theory. We focus on joint-action learning, so as to restrict ourselves to review only techniques that combine reinforcement learning driven by game-theoretical advances.

## 2 Markov Decision Processes

Markov Decision Processes (MDPs) provide the mathematical framework for modeling decision making with single agents operating in a fixed environment. Therefore, we do not explicitly model secondary agents, which can also be viewed as part of the environment.

A Markov decision process is defined by a tuple $\{S, A, T, R\}$ where

- S is the set of states
- A is the set of actions
- $T : S \times A \rightarrow p(S)$ is the transition function that defines the probability distribution over the next states as a function of the current state and the action taken
- $R : S \times A \rightarrow \mathcal{R}$ is the reward function which determines the reward received by the agent as a result of choosing an action in a given state

The agent's objective is to find the optimal strategy $\pi$ that maximizes his future expected reward (expected sum of discounted reward)

$$V^{\pi}(S) = \mathbb{E}[\sum_{j=0}^{\infty} \gamma^{j} r_{t+j}] \tag{1}$$

where $r_{t+j}$ is the reward received j steps into the future and $0 \leq \gamma \leq 1$ is a discount factor that models the importance of future rewards. A discount factor close to 0 will result in an agent driven mostly by short-term rewards, while an agent with discount close to 1 will have an optimal policy that works towards long-term high reward.

The value $V^{\pi}(s)$ of a state given a policy is the expected return starting from state s and following the policy, while the state-action value $Q^{\pi}(s,a)$ is the expected return starting from state s, taking action a and follow the policy afterwards. By computing these values, we can find the optimal policy.

Every Markov decision process has at least one stationary, deterministic optimal policy $\pi^* : S \rightarrow A$; stationary means that the policy does not change as a function of time, while deterministic means that the same action is always chosen in state s for all s $\in S$. Optimal means that the policy is undominated: there is no state from which any other policy can achieve a higher expected sum of discounted reward.

Algorithms for solving MDPs are clustered into two groups with respect to the transition function. When we assume that the transitions from a state to another state are known and given, we are dealing with a Dynamic-Programming setting, where planning is our main focus and convergence is guaranteed [3]. Thus, algorithms such as Value iteration or Policy iteration are included in this category. On the contrary, when the transitions are unknown, we are in the Reinforcement Learning (RL) setting, with algorithms that require online updates or sampling of sequences from the MDP to solve the optimization problem. One of the most popular algorithms in RL is Q-learning. We will briefly mention Value iteration and Q-learning, as we will see how they can be extended to the multi-agent case. We refer the reader to [13] for a detailed view of RL.

### 2.1 Value Iteration

In Value Iteration [1] we start at time step $t = 0$ with a random initialization of the value function and then repeatedly compute $V_{t+1}$ for all states s using the equations (2) and (3) until convergence.

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s') \tag{2}$$

$$V(s) = \max_{a' \in A} Q(s,a') \tag{3}$$

The optimal policy can be computed as $a^* = \underset{a' \in A}{\operatorname{argmax}} Q(s,a')$.

### 2.2 Q-learning

When the transition function is unknown, an update is performed by an agent whenever it receives reward r when making transition from s to s' after taking action a.

$$Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)) \tag{4}$$

or equivalently

$$Q(s,a) \leftarrow (1-\eta)Q(s,a) + \eta(r + \gamma V(s')) \tag{5}$$

where $\eta$ is the learning rate (usually initialized to 1.0 and decays over time)

Q-learning [16] is proved to converge if every action is tried in every state infinitely often and new estimates are blended with previous ones using a slow enough exponentially weighted average [15]

We continue with the framework for Markov Games and algorithms for MARL.

## 3 Stochastic Games

Stochastic or Markov games were proposed as the standard framework for modeling multiple adaptive agents with interacting or competing goals [9]. Markov games still assume that state transitions are Markovian, however the difference with the single-agent MDP framework is that each agent has its own set of actions. For n agents, the joint-action space is $A = A_1 \times A_2 \times \ldots \times A_n$. The state transition $T : S \times A_1 \times \ldots \times A_n \rightarrow p(S)$ and reward functions $R_i : S \times R_1 \times \ldots \times R_n \rightarrow \mathcal{R}$ now depend on the joint action of all agents. Similar to the MDP objective, agent i is trying to maximize his expected reward under a joint policy $\pi = (\pi_1, \ldots, \pi_n)$, which assigns a policy $\pi_i$ to each agent i:

$$V_i^{\pi}(s) = \mathbb{E}[\sum_{j=0}^{\infty} \gamma^j r_{i,t+j}] \tag{6}$$

Contrary to the MPD case, there may not exist an optimal stationary deterministic policy, meaning that the optimal stationary policy is sometimes probabilistic; mapping states to discrete probability distributions over actions $\pi^* : S \rightarrow p(A)$. The best response and Nash equilibrium concepts can be extended to such games: a policy $\pi_i$ is the best response if there is no other policy for agent i that gives higher expected future reward when other agents keep their policies fixed.

If we assume only 1 agent, or the case where other agents play a fixed policy, the Markov game reduces to an MDP.

When the Markov game has only 1 state, it reduces to a repeated normal form game, a common benchmark for multi-agent learning in which players simultaneously select an individual action to perform and receive a reward based on their joint action, after which the game ends. Thus, there is no state transition function and there is no delayed reward, which is one of the most essential properties of RL. However, since we are considering other agents' behavior (policies), we are still dealing with non-stationary environments. Methods in this particular category turn efforts towards adapting policies that depend on the actions of other agents.

## 4 Algorithms for General Markov Games

### 4.1 Assumptions

In the general RL setting, which is also applied here, we assume that agents do not have access to the reward function or are aware of the expected reward

from playing a joint action. In addition to this, we also assume that joint actions do not produce the same deterministic reward for each agent, i.e. payoffs are stochastic. Sampling these joint actions repeatedly is a necessity.

In contrast to the general RL methods, we also make assumptions regarding the observations that agents make. Due to the dependence on other agents' strategy, most MARL methods assume that either actions of all participating agents or rewards (or both) are observed from all agents. Although this help us model opponents and learn over joint actions, it may be unrealistic to assume such common knowledge, especially in distributed systems where such information is usually not be available.

A major division of algorithms for MARL is whether we are considering other agents as part of the environment (Independent Learning) or we explicitly try to model other agents (Joint Action Learning). We begin with a brief description of the first setting and then we continue with a description of several algorithms for the second one, which is the main focus of this work.

### 4.2 Independent Learning

As mentioned above, here we reduce the multi-agent case to a single-agent learning problem where interaction with other agents is implicitly perceived as noise. Traditional reinforcement learning algorithms, such as SARSA or Q-learning can be used in this case, however convergence guarantees are no longer available due to the stochasticity in the environment from the other agents. Moreover, there is no mechanism for coordination, which is particularly useful for decentralized systems.

### 4.3 Joint Action Learning

In this setting we explicitly consider other agents by learning in the space of joint actions. Thus, the framework becomes appropriate for modeling coordination among agents, but the algorithmic complexity grows exponentially with the number of agents. Another disadvantage that was previously described is that an agent has to make assumptions regarding the opponents' strategies to predict their future actions.

We provide a description for most of the algorithms that extend the Q-learning for multi-agent RL. Here, the agent estimates Q(s,$\mathbf{a}$) for taking the joint action $\mathbf{a} = a_1, \ldots, a_n$ in state s. Since the actions of other agents can vary, the agent does not have one estimate for the reward he will receive by taking action $a_i$ in state s, but instead he has to keep estimates for each combination of his action $a_i$ and the joint action $a_{-i}$ played by the other agents. We therefore have to calculate the value of a state by taking into account the actions of the other agents. It is easy to see how the complexity increases by the number of agents involved and the number of possible actions for each agent.

### 4.4 Joint Action Learning (JAL)

In this algorithm we estimate other agents' policies to determine the expected probability with which each different joint actions is played. This probability distribution of joint actions can then be used to calculate the value of a state. In the Joint Action Learner (JAL) algorithm [4], an agent counts the number of times a joint action pair is played by the other agents in a given state, denoted by $c(s, a_{-i})$. A normalized version of the frequencies is used to calculate the value of a state:

$$V_i(s) = \max_{a_i \in A_i} Q(s, a_i) = \sum_{a_{-i} \in A_{-i}} \frac{c(s, a_{-i})}{\sum_{a_{-i'} \in A_{-i}} n(s, a_{-i'})} Q(s, a_i, a_{-i}) \tag{7}$$

where $A_{-i}$ is the possible action set for all other agents and $Q(s, a_i, a_{-i})$ is the Q-value in state s for the joint action in which agent i plays $a_i$ and other agents play a joint action $a_{-i}$. All other aspects of the Q-learning update remain the same as in the standard single-agent Q-learning algorithm. The advantage of this method is the low additional complexity, which only comes from storing and updating these counts. However, the algorithm works well only for deterministic tasks, where all agents converge in the end in choosing the same action in a given state s.

### 4.5 Minimax-Q

The minimax-Q algorithm [9] was developed for two-player zero-sum games and gives a preference in conservative strategies by employing the minimax principle ("behave so as to maximize your reward in the worst case"). By using the minimax principle, we assume that an agent's opponent will play the action which minimizes the agents payoff and as a result we try to maximize the optimal agent's minimum expected reward.

In the case of two-player zero-sum games, for the strategy $\pi^*$ to be optimal it needs to satisfy

$$\pi^* = \operatorname*{argmax}_{\pi \in p(A)} \min_{o \in O} \sum_{a \in A} R_{o,a} \pi_a \tag{8}$$

where $R_{o,a}$ reward for agent taking action a and opponent taking action o.

Both Value Iteration and Q-learning, as well as other algorithms can be easily extended by replacing the max operator of single agent Q-learning with the minimax value.

For example, Value Iteration becomes:

$$Q_i(s, a, o) = R_i(s, a, o) + \gamma \sum_{s' \in S} T(s, a, o, s') V_i(s') \tag{9}$$

$$V_i(s) = \max_{\pi \in p(A)} \min_{o \in O} \sum_{a \in A} Q_i(s, a, o) \pi_a \tag{10}$$

$$\pi_i^* = \underset{\pi \in p(A)}{\operatorname{argmax}} \min_{o \in O} \sum_{a \in A} Q_i(s, a, o)\pi_a \tag{11}$$

Moreover, Q-learning extends to minimax-Q

$$Q_i(s, a, o) \leftarrow (1 - \eta)Q_i(s, a, o) + \eta(r + \gamma V_i(s')) \tag{12}$$

$$V_i(s) = \min_{o' \in O} \sum_{a' \in A} \pi_i^*(s, a')Q_i(s, a, o') \tag{13}$$

$$\pi_i^*(s, a) \leftarrow \underset{\pi_i}{\operatorname{argmax}} \min_{o' \in O} \sum_{a' \in A} \pi_i(s, a')Q_i(s, a', o') \tag{14}$$

The optimization problem in (14) can be solved using linear programming. Even if the minimax optimization has multiple solutions, any of them will achieve at least the minimax return regardless of what the opponent is doing. Moreover, it is shown that when the opponent is suboptimal (does not always chooses the optimal action) we might get better results.

Convergence is guaranteed only for two-player zero-sum games and assuming that the other agent executes all of its actions infinitely often. However, it provides an opponent-independent method for learning an equilibrium solution, since convergence also holds even when the other agent does not converge to the equilibrium. On the other hand, minimax is very conservative, as it tries to maximize the worst-case performance. In some cases it might be more beneficial to adjust the risk taken to maximize the reward w.r.t. the opponent's behavior.

### 4.6 Nash-Q

Hu et al. [6, 7] extended the Minimax-Q algorithm to general-sum games. Here each agent maintains Q values for all the other agents. A Nash-Q agent assumes that all agents will play according to a Nash equilibrium in each state:

$$Q_i(s, a_1, \ldots, a_n) \leftarrow (1 - \eta)Q_i(s, a_1, \ldots, a_n) + \eta(r + \gamma V_i(s')) \tag{15}$$

$$\text{where } V_i(s) = Nash_i(s, Q_1(s, a_1, \ldots, a_n), \ldots, Q_n(s, a_1, \ldots, a_n)) \tag{16}$$

$$Nash_i(s, Q_1(s, a_1, \ldots, a_n), \ldots, Q_n(s, a_1, \ldots, a_n)) = \pi_1(s) \ldots \pi_n(s)Q_i(s', a_1, \ldots, a_n) \tag{17}$$

$Nash_i(s, Q_1(s, a_1, \ldots, a_n), \ldots, Q_n(s, a_1, \ldots, a_n))$ is the expected payoff for agent i when the agents play a Nash Equilibrium in state s with Q-values $Q_1, \ldots, Q_n$

While Minimax-Q uses linear programming to solve the optimization problem for zeros-sum games, Nash-Q uses quadratic programming to find an equilibrium in general-sum games. Nash-Q keeps the opponent-independence property of Minimax-Q, but one major assumption due to this independence is that the game must have a unique equilibrium, which is not always true for general-sum stochastic games. When multiple equilibria exist, the agents should agree to play the same equilibrium, which again is not always possible.

The algorithm converges to Nash-Q values if either every stage game (one-period game) encountered by the agents during learning has a global optimum point, and the agents update according to values at this point, or if every stage game has a saddle point, and agents update in terms of these[6, 7]. This requirement is satisfied only in special cases, as properties of stage games during learning are difficult to ensure in general. Therefore extending to external mechanisms for equilibrium selection could be useful for convergence.

### 4.7 Correlated-Q

Greenwald et al. [5] extend Nash-Q to include correlated equilibria instead of Nash equilibria, so as to solve the case where coordination is needed to agree on the same equilibrium.

$$V_i(s) = CE_i(s, Q_1(s, a_1, \ldots, a_n), \ldots, Q_n(s, a_1, \ldots, a_n)) \tag{18}$$

where $CE_i(s, Q_1(s, a_1, \ldots, a_n), \ldots, Q_n(s, a_1, \ldots, a_n))$ is the agent i's reward according to some correlated equilibrium in the general-sum game determined by the Q-values $Q_1, \ldots, Q_n$

The authors additionally introduce four variants of correlated-Q learning, based on four correlated equilibrium selection mechanisms to address the difficulty in the equilibrium selection problem.

### 4.8 Friend-or-Foe Q-learning

In this setting other agents are classified into two groups: friends (coordination) with updates similar to Q-learning or foes (opponents) with updates similar to minimax-Q [8].

Although defined for any number of players, the authors show the updates for the two-player game:

$$\text{Friend: } V_1(s) \leftarrow \max_{a_1 \in A_1, a_2 \in A_2} Q_1(s, (a_1, a_2)) \tag{19}$$

$$\text{Foe: } V_1(s) \leftarrow \max_{\pi_1 \in \Pi(A_1)} \min_{a_2 \in A_2} \sum_{a_1 \in A_1} \pi_1(a_1) Q_1(s, (a_1, a_2)) \tag{20}$$

Littman [8] proves that Friend-or-Foe Q-learning converges in general-sum Markov games, however equilibrium policies are restricted in these two classes of games.

## 5 Drawbacks of MARL and Future Work

One disadvantage or RL is the exponential growth of the state-action space in the number of state and action variables (curse of dimensionality [13]). MARL faces additional complexity, exponential in the number of agents. Learning in the joint state-action space makes the above approaches feasible only for small

environments and with a limited number of agents. One way to address this is to consider modeling other agents only when a better payoff can be obtained by doing so, and ignore other agents otherwise. This reduces the state-action space while providing a way to deal with agents when it is beneficial. An example of such systems is an Robocup, a soccer game player by robots or autonomous cars, where agents can consider each other when they are in close proximity. However, usually most MARL algorithms are evaluated on small problems and it seems unlikely that we could scale up in larger or continuous state and action spaces.

Non-stationarity is also an issue, since the best policy changes as other agents' policies change. Therefore, convergence in the general Markov game case remains an unsolved theoretical problem.

Moreover, limited research has been done in the case of incomplete or uncertain observations as well as in extending to unknown state space. Partial observable states or games with incomplete information [11] include modeling belief over the state variable, which can become extremely complicated since the state-space becomes unbounded and we have to deal with nested beliefs (beliefs that players hold about each others beliefs). A first approach applied to finite-horizon very small POSGs combines policy iteration and iterative elimination of dominated strategies (policies) [2]. Furthermore, an approximate probabilistic inference approach using the EM algorithm [12] empirically showed convergence on a variety of two-player games but there is no known tractable solution for computing optimal policies in general Partially Observed Stochastic Games (POSGs).

Finally some researchers argue that Q-values may not be sufficient to learn an equilibrium policy in arbitrary general sum games [17]. Recently, methods from evolutionary game theory (which seems related to the concepts of Genetic Algorithms[1] and Neuroevolution[2]) have been successfully employed to multi-agent learning [14], however this remains as a future exploration task that is not the main purpose of this review.

Given this recent change in direction and the algorithms described in this survey, we believe that advances in multi-agent research can be achieved by a more thorough combination of machine learning and game theory techniques.

[1] `https://en.wikipedia.org/wiki/Genetic_algorithm`
[2] `https://en.wikipedia.org/wiki/Neuroevolution`

## References


[1] Richard Bellman. *Dynamic Programming*. 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[2] Daniel S Bernstein et al. "Policy iteration for decentralized control of Markov decision processes". In: *Journal of Artificial Intelligence Research* 34.1 (2009), p. 89.

[3] D Bertsekas. "Dynamic Programming: Deterministic and Stochastic Models Prentice-Hall". In: *Englewood Cliffs, NJ* (1987).

[4] Caroline Claus and Craig Boutilier. "The dynamics of reinforcement learning in cooperative multiagent systems". In: *AAAI/IAAI*. 1998, pp. 746–752.

[5] Amy Greenwald, Keith Hall, and Roberto Serrano. "Correlated Q-learning". In: *ICML*. Vol. 3. 2003, pp. 242–249.

[6] Junling Hu and Michael P Wellman. "Nash Q-learning for general-sum stochastic games". In: *The Journal of Machine Learning Research* 4 (2003), pp. 1039–1069.

[7] Junling Hu, Michael P Wellman, et al. "Multiagent reinforcement learning: theoretical framework and an algorithm." In: *ICML*. Vol. 98. Citeseer. 1998, pp. 242–250.

[8] Michael L Littman. "Friend-or-foe Q-learning in general-sum games". In: *ICML*. Vol. 1. 2001, pp. 322–328.

[9] Michael L Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *Proceedings of the eleventh international conference on machine learning*. Vol. 157. 1994, pp. 157–163.

[10] Jiyun Luo, Xuchu Dong, and Hui Yang. "Learning to Reinforce Search Effectiveness". In: *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*. ACM. 2015, pp. 271–280.

[11] Liam MacDermed, Charles Isbell, and Lora Weiss. "Markov games of incomplete information for multi-agent reinforcement learning". In: *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*. 2011.

[12] Tim Rakowski et al. "Inference-based Decision Making in Games". In: (2011).

[13] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Cambridge Univ Press, 2011.

[14] Karl Tuyls and Simon Parsons. "What evolutionary game theory tells us about multiagent learning". In: *Artificial Intelligence* 171.7 (2007), pp. 406–416.

[15] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[16] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". PhD thesis. University of Cambridge England, 1989.

[17] Martin Zinkevich, Amy Greenwald, and Michael Littman. "Cyclic equilibria in Markov games". In: *Advances in Neural Information Processing Systems* 18 (2006), p. 1641.