RESEARCH ARTICLE

Learning Multirobot Hose Transportation and Deployment by Distributed Round-Robin Q-Learning

Borja Fernandez-Gauna[1], **Ismael Etxeberria-Agiriano**[2], **Manuel Graña**[3]*

1 Polytechnical Institute and Computational Intelligence Group, University of the Basque Country (UPV/EHU), Vitoria-Gasteiz, Spain, **2** Polytechnical Institute, University of the Basque Country (UPV/EHU), Vitoria-Gasteiz, Spain, **3** Computational Intelligence Group, University of the Basque Country (UPV/EHU), San Sebastian, Spain, & ENGINE centre Wroclaw University of Technology (WrUT), Wroclaw, Poland

* manuel.grana@ehu.eus

OPEN ACCESS

Citation: Fernandez-Gauna B, Etxeberria-Agiriano I, Graña M (2015) Learning Multirobot Hose Transportation and Deployment by Distributed Round-Robin Q-Learning. PLoS ONE 10(7): e0127129. doi:10.1371/journal.pone.0127129

Editor: Catalin Buiu, Politehnica University of Bucharest, ROMANIA

Received: July 28, 2014

Accepted: March 14, 2015

Published: July 9, 2015

Data Availability Statement: Source code is available at https://github.com/borjafdezgauna/D-RR-QL-PlosONE.

Funding: This research has been partially funded by EU through SandS project, grant agreement no 317947. This research has been partially funded by grant TIN2011-23823 of the Ministerio de Ciencia e Innovación of the Spanish Government (MINECO), with FEDER funds. The GIC has been supported by grant IT874-13 as university research group category AMG was supported by EC under FP7, Coordination and Support Action, Grant Agreement Number 316097, ENGINE European Research Centre of

Abstract

Multi-Agent Reinforcement Learning (MARL) algorithms face two main difficulties: the curse of dimensionality, and environment non-stationarity due to the independent learning processes carried out by the agents concurrently. In this paper we formalize and prove the convergence of a Distributed Round Robin Q-learning (D-RR-QL) algorithm for cooperative systems. The computational complexity of this algorithm increases linearly with the number of agents. Moreover, it eliminates environment non sta tionarity by carrying a round-robin scheduling of the action selection and execution. That this learning scheme allows the implementation of Modular State-Action Vetoes (MSAV) in cooperative multi-agent systems, which speeds up learning convergence in over-constrained systems by vetoing state-action pairs which lead to undesired termination states (UTS) in the relevant state-action subspace. Each agent's local state-action value function learning is an independent process, including the MSAV policies. Coordination of locally optimal policies to obtain the global optimal joint policy is achieved by a greedy selection procedure using message passing. We show that D-RR-QL improves over state-of-the-art approaches, such as Distributed Q-Learning, Team Q-Learning and Coordinated Reinforcement Learning in a paradigmatic Linked Multi-Component Robotic System (L-MCRS) control problem: the hose transportation task. L-MCRS are over-constrained systems with many UTS induced by the interaction of the passive linking element and the active mobile robots.

Introduction

The transportation problem The transportation of a hose by a team of robots is a paradigmatic instance of the tasks that can be performed with *Multi-Component Robotic Systems* (MCRS) [1]. In fact, a hose with a team of robots attached to it is a Linked MCRS (L-MCRS), because it can be viewed as a collection of autonomous robots physically connected by a passive two-dimensional object, i.e. the hose.

Network Intelligence for Innovation Enhancement. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing Interests: The authors have declared that no competing interests exist.

Autonomous learning Reinforcement Learning (RL) [2, 3] is the main paradigm for autonomous learning. Agent-environment interaction is modeled as a Markov Decision Process (MDP) specified by a system state space, the transitions between states, and the actions that the agent can perform in each system state. Agent learning is guided by an external *reward* signal that gives a feedback assessment of the value of an action executed while the system is at some state. The goal of RL is to estimate optimal action selection policies maximizing the expected reward. In most cases, the agent-environment MDP is not an irreducible ergodic Markov Chain, but has some terminal states where the system lies forever if reached, i.e. probability of exiting the state is zero regardless of the action taken. While learning, the system must be re-initialized after reaching the terminal state in order to continue learning and exploration. Therefore, the overall learning process carried by RL is composed of successive trial episodes, which are independent realizations of the evolving MDP starting from (random) initial states, i.e. the MDP changes some of its policy making parameters in between or during trials. The simplest model-free RL algorithm is Q-learning [4, 5], which applies an iterative reward propagation rule to estimate the state-action value function implementing the optimal policy. It is often the case that rewards only happen in succesful termination states, so that learning involves the repetition of the task trial many times. Autonomous learning of the L-MCRS control by single-agent RL approaches has been demostrated [6, 7] in the small scale cases, however applying RL to systems with many robots faces an exponential computational complexity growth. Therefore, we are looking for distributed multi-agent approaches which promise computational speed-up through parallel processing, direct modeling of multi-component systems, fault-tolerance inherent to distributed control realizations, and the ability to provide linear complexity approximations to problems which are combinatorial in nature, so that their complexity grows exponentially with the number of agents.

Overconstrained systems In robotic applications, normal termination is given by the accomplishment of the assigned task, so that the agent obtains a positive reward. Undesired terminal states (UTS) correspond to irreversible situations where the system is stuck in a rewardless state and no evolution to a successful termination of the proposed task is possible. Hence, the work performed to reach this state is fruitless, the agent does not extract any positive training information. UTS often correspond to catastrophic error conditions. We say that a system is over-constrained when the number of UTS is large relative to the state space size. In over-constrained systems, RL convergence is severely handicapped by the high frequency of learning episodes ending in UTS. In single-agent domains, Modular State-Action Vetoes (MSAV) [8] provide a convergence speedup by early avoidance of UTS, minimizing unnecessary computation for the estimation of state-action values. The reward signal is decomposed into separate signal classes, each of them associated with a particular class of physical constraint which is commonly related to a specific subset of state variables. Modules specialized in each signal class can be taught independently when there is no interaction between state variables modeling the failure signal classes.

MARL Multi-Agent Reinforcement Learning (MARL) [9] scenarios involve several agents learning concurrently, so that some coordination mechanism is required for agents to agree on the joint-action to be taken. MARL has been successfully applied to several problems, such as traffic control [10], supply chain ordering management [11], prey chase [12], or intrusion detection [13]. The concurrent adaptation of the individual agent policies makes the environment non-stationary from an agent's perspective. For this reason, few MARL algorithms offer any theoretical results about convergence to an optimal joint-policy. Moreover, solving the coordination problem ensuring convergence has additional cost in memory or communication resources [14]. The main issue in Q-Learning based MARL approaches is the exponential growth of the state-action space when the number of agents is increased (*curse of* *dimensionality*), because it needs to consider all possible combinations of local agent actions, even when all agents share a common set of state variables. This problem has been dealt with using general Function Approximators which reduce the amount of information storage needed [15–21], but data fitting processes impose specific assumptions which can bias the solution [17, 22]. Besides, most MARL approaches need communication between agents either for coordination or for internal computations. Communication channels introduce delays and noise compromising the system's learning convergence. Depending on the nature of the rewards, MARL systems can be cooperative or competitive [9]. In cooperative systems, all agents receive the same reward, perceived as the result of the joint actions of the agents in the actual system state without requiring explicit communication between agents. We will only consider cooperative systems.

Paper contribution This paper formalizes and proves the convergence of the *Distributed Round-Robin Q-Learning* (D-RR-QL) algorithm in which agents follow a predefined Round Robin (RR) order to execute their local actions sequentially. Agents are endowed with local Q-tables performing local estimation of the state-action value function. Local agent policies are composed into a joint policy that approximates the global optimal policy using a message-based procedure. D-RR-QL converges to an optimal policy even in stochastic environments. Besides, sequential execution of local actions allow agents to use MSAV when dealing with over-constrained systems, because the outcome depends only on the local action. Computational experiments show that in over-constrained environments, D-RR-QL with MSAV is able to estimate a good approximation to an optimal policy, whereas competing state-of-the-art algorithms D-QL [23], Team QL [24], and Coordinated RL [25] never reached the goal state, even when allowed a bigger learning time span, so that they were unable to perform any learning step.

Problem Statement

The specific instance of the hose transportation problem dealt with in this paper is represented in Fig 1. A set of robots is attached to a hose and they must maneuver so that the tip of the hose reaches a predefined goal destination. The hose is a passive linking element that constrains non-linearly the motion of the robots. The dynamics of the whole system are highly non-linear.

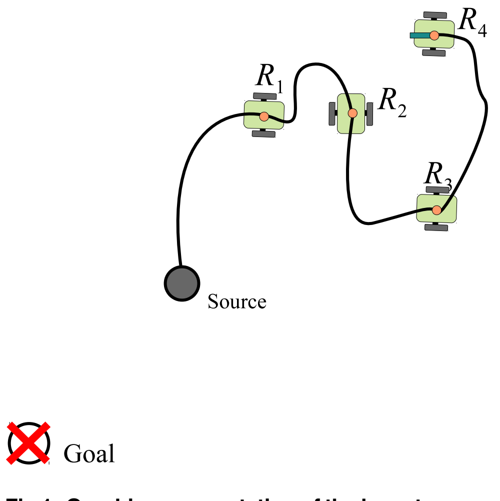

Fig 1. Graphic representation of the hose transportation problem. The robots attached to the hose must move coordinately so that the tip of the hose carried by the robot R_4 reaches the goal position.

doi:10.1371/journal.pone.0127129.g001

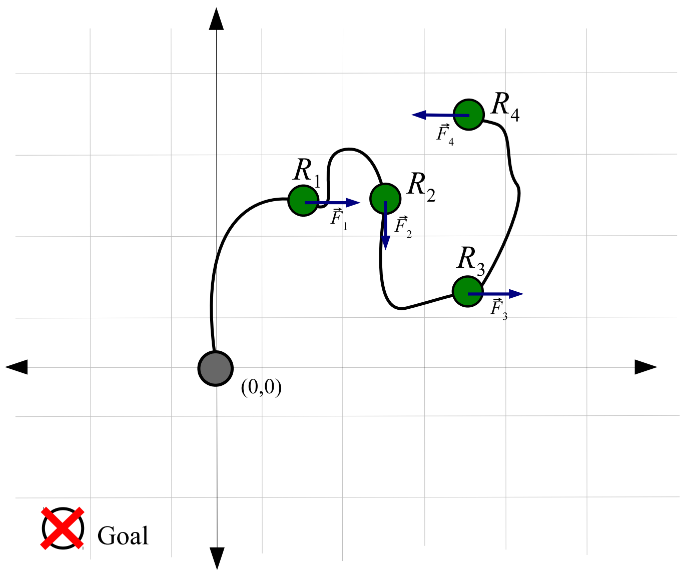

Fig 2. Second order model approach to the hose transportation problem applying GEDS. Arrows indicate the actual force applied by the robots on the hose.

doi:10.1371/journal.pone.0127129.g002

Accurate dynamical modeling of the hose motion pushed by the robots can be achieved using bi-dimensional Geometrically Exact Dynamic Splines (GEDS) [26–28]. The spline is defined as $\mathbf{q}(u,t) = \sum_{nc}^{i=0} N_{i,d}(u) \cdot \mathbf{c}_i(t)$, where $\mathbf{c}_i(t)$, $i = 1, \ldots, nc$ are the dynamic control points and $N_{i,\,d}$ is the d-degree polynomial specified by control point $c_i(t)$. This equation defines the position of the spline for a normalized value $u \in [0,1)$ at time t. Robots are modeled as control points along the spline, and the actions executed by robots are modeled as the external forces $\vec{F}_i,\ i = 1, \ldots, n$ exerted on the hose. The GEDS model is illustrated in Fig 2.

Simulation of the linked system using GEDS modeling is computationally very expensive (for some systems it takes about 2 minutes to simulate a single time step on a standard desktop computer). Transfer learning can be applied to overcome this computational cost barrier, i.e. tackling the problem as a two-step process: first, agents are trained on a simplified model that uses line segments to model the passive linking element, such as in Fig 3, so that they acquire the basic control skills through RL; second, this knowledge is transferred to learning the control of the robots when the hose is modeled by a GEDS, so that agents refine their control skills with the most accurate model [29].

Most important, the hose-robots system is an overcontrained system. Whenever any of the dynamic constraints is broken the system reaches a UTS, and it must be reset so that the learning episode needs to start again from the initial setting. Specifically, we consider four dynamic constraints: (a) the robots are not allowed to step over the hose, (b) the hose cannot be overstretched, (c) robots must not collide with each other, and (d) they must stay within the simulation bounds.

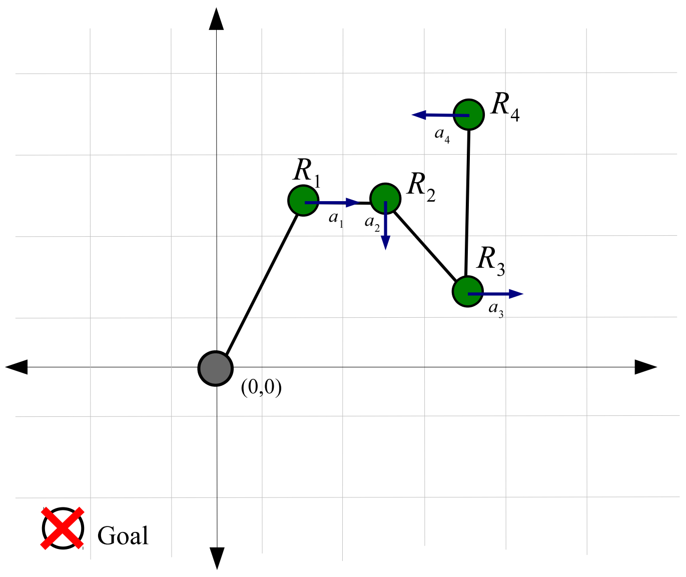

Fig 3. First order model approach to the hose transportation problem using line segments to represent the hose. Arrows indicate the actual force applied by the robots on the hose.

doi:10.1371/journal.pone.0127129.g003

Reinforcement Learning Background

Single-Agent Reinforcement Learning

Markov decision processes. In RL algorithms, the interaction between an agent and its environment is modeled as a Markov Decision Process (MDPs), defined as a tuple $< S, A, P, R >$, where S is a finite set of states, A is a set of actions which the agent can execute, $P{:}S \times A \times S \to [0, 1]$ is a probabilistic transition function $P(s, a, s')$ giving the probability of observing state s' after executing action a in state s, and $R{:}S \to \mathbb{R}$ is the expected immediate reward after reaching state s. The set of terminal states $\mathcal{T}$ is defined as the set of states having null outgoing transition probabilities, i.e. $\mathcal{T} = \{s_S | \forall s' \neq s, a; P(s, a, s') = 0\}$. The action selection policy is usually modeled as the probability distribution $\pi{:}S \times A \to [0, 1]$ of taking action a in state s.

The *value* $V^\pi(s)$ of state s is the expected accumulated reward obtained from that state following policy π. It can be expressed as

$$\begin{aligned} V^\pi(s) &= E^\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \\ &= \sum_a \pi(s, a) \sum_{s'} P(s, a, s')[R(s') + \gamma V^\pi(s')], \end{aligned} \tag{1}$$

where E^π denotes the expected return from time step t following policy π thereafter, r_i represents the reward received at time i, and $\gamma \in [0, 1]$ is a discount-rate parameter decreasing the weight of the reward at each time-step. Eq (1) is the *Bellman equation for* V^π [3]. There always exists one or more optimal policies π^* maximizing the state value, denoting V^* this maximum value. It satisfies:

$$V^*(s) = \max_{a \in A} \left\{ \sum_{s'} P(s,a,s')[R(s') + \gamma \cdot V^*(s')] \right\}. \tag{2}$$

The goal of the learning agent is to find this optimal policy. The expected value of executing an action a in state s following a policy π, is modeled by the state-action value function $Q^{\pi}(s, a)$:

$$Q^{\pi}(s,a) = E^{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\},$$

where a_t represents the action taken at time-step t. The optimal state-action value function is obtained applying an optimal policy π^*:

$$Q^*(s,a) = \sum_{s'} P(s,a,s') \left\{ R(s,a,s') + \gamma [\max_{a'} Q^*(s',a')] \right\}. \tag{3}$$

An episode consists of a sequence of state transitions fired by actions executed by the agent, leading to a terminal state that represents the success/failure of the completion of a proposed task. The deterministic *greedy* action selection involves selecting always the action with the highest Q-value *exploiting* the available knowledge

$$\pi_{greedy}(s) = \arg\max_{a' \in A} Q(s,a'),$$

implementing the optimal policy implied by the actual Q-table. This policy is applied in the test and operational phases of the system. In order to ensure convergence during learning, the system must be able to explore, i.e. to reach any state from any state, therefore action selection probabilities must be non-null. One way to ensure that is an stochastic action selection algorithm such as *Soft-Max*:

$$\pi_{SoftMax}(s,a,\tau) = \frac{\exp^{Q(s,a)/\tau}}{\sum_{a'} \exp^{Q(s,a')/\tau}}, \tag{4}$$

where $\tau \in (0,1]$ is a parameter controlling the flatness of the probability distribution, aka temperature parameter.

Q-Learning. Q-Learning [5] is a model-free Temporal Difference reinforcement learning algorithm performing an iterative estimation of the state-action value function $Q(s, a)$, hence its name. It allows agents to find the optimal policy for an MDP without *a priori* knowledge about the transition function P and the reward function R. Denote $\{Q_t(s, a); t = 0, 1, 2, \ldots\}$ the sequence of state-action value function estimations during RL, converging to $Q^*(s, a)$.

The original Q-Learning algorithm stores in tabular form state-action values $Q(s, a)$. At each learning episode, the agent observes the actual state s_t, selects an action to be executed a_t, receives the immediate reward r_t, observes the subsequent state s' and updates the state-action value matrix to $Q_t(s, a)$ from the previous estimation $Q_{t-1}(s, a)$. Learning speed is controlled by the gain α_t and Q-values are updated using the following rule:

$$Q_t(s,a) \leftarrow (1-\alpha_t) Q_{t-1}(s,a) + \alpha_t \left[r_t + \gamma \max_{a'} Q_{t-1}(s',a') \right] \tag{5}$$

Q-learning converges with probability 1 to the optimal policy in a stationary environment [5] when all system's states are visited infinitely often, and the learning gain α_n complies with the stochastic gradient conditions. This ergodic condition on the stochastic process realized by the agent according can not be fulfilled by an MDP containing terminal states ($\mathcal{T} \neq \emptyset$) or having sparse state transition probability matrix. To avoid the latter, SoftMax action selection function is applied. To overcome the former, the learning process must be reinitialized each time that it falls in a terminal state. A separate representation for each state-action value is also required, and thus, the storage space requirements are $O(|S \times A|)$, growing linearly with the state-action product space size.

Multi-agent Reinforcement Learning

MARL algorithms were classified in [30] depending on agents' perception capabilities: in *Independent Learners* (IL) approaches, agents are able to observe the global state but restricted to know their own local actions, while in *Joint-Action Learners* (JAL) approaches, agents are allowed to observe both the global state and all the actions taken by all agents. IL are a more realistic framework for real life deployment of MCRS systems, because communication requirements are inmediately scalable.

Independent Learners (IL). IL approaches decompose the global state-action value Q-table into local Q-tables owned by agents. The easliest instance of this approach is the naive IL approach [30], where agents implement a single-agent RL algorithm obviating the decisions and the payoff obtained by the remaining agents. Because of the lack of a coordination mechanism, the system cannot be guaranteed to converge to a stable or a globally optimal policy. The *Distributed Rewards* and *Distributed Value Functions* were studied in [31] as a way to motivate cooperation between neighbors. Instead of updating the state-action values using only the local reward or values, agents also used weighted rewards or state-action values of their teammates. The computational complexity of these methods scale linearly with the number of agents, but no proof of convergence exists. *Distributed Q-Learning* (D-QL) [23] performs local training on a local Q-table *per* agent assuming optimal behavior from all remaining agents. The virtual global Q-table is decomposed into local Q-tables. For each local action a_i, the agent stores only the value of the joint-action containing a_i that maximizes the reward. Thus, agents need not be aware of the actions taken by the rest of agents to be able to converge to a globally optimal policy. However, this convergence is not guaranteed in stochastic environments. Some authors propose the use of an actual centralized table updated by all agents [32, 33]. Some others have presented very good results applying a heuristically modified version of D-QL, known as Hysteretic QL [34]. Distributed RL has also been proposed to deal with multi-objective optimization problems using negotiation protocols between agents [35].

Joint-Action Learners (JAL). The most straight-forward JAL approach to MARL is to implement an independent Q-Learning process in each agent estimating the Q-value of each of the joint-actions. *Team Q-Learning* [24] (Team-QL) assumes that a unique optimal action decision exists in each state.

Some authors have proposed model based heuristic algorithms [30, 36] to estimate the most likely response of the remaining agents, using them to bias local policies towards coordinated joint actions. Those models are learned from experience. Following a different approach, each state in an MDP can be regarded as a virtual stateless Stochastic Game (SG), therefore adaptive methods [14, 37] have been proposed to bias local action selection towards a globally optimal joint action. These approaches require additional memory resources and knowledge about the optimal state-joint-action function, scaling badly with the problem size.

The *Coordinated Reinforcement Learning* (Coordinated-RL) [25] approximates the global joint value function as a linear combination of local value functions [38]. The complexity of agreeing on a globally optimal joint action can be reduced assuming that agents need not coordinate with all remaining agents, but with a small subset. These coordination dependencies between agents are context-specific, can change dynamically and can be defined as a *Coordination-Graph*, where undirected edges represent a coordination dependence between agents. The use of the Coordination Graph reduces the state-action space by defining which actions are relevant to each local value function, and it can still be further reduced by identifying which state variables are relevant to each local value function.

Related Work

In this section, we first review previous work on dynamic constraints in single-agent RL. To the best of our knowledge, no relevant work can be found on constraints in multi-agent systems. Next, we describe the three MARL algorithms that we have used in our experiments for benchmarking: Team Q-Learning, Distributed Q-Learning, and Coordinated RL.

Dynamic Constraints in RL

In the literature about RL, undesired terminal states (UTS) have been dealt with in different ways: (a) Using null state transitions and/or generating negative rewards [39–41]. (b) Manually discarding actions that could lead to an undesirable state [42], defining a state dependent action repertoire $A(s)$. (c) Defining MDPs with Constraints [43–45], where the goal is to maximize the expected accumulated reward subject to minimize the risk of ending in an *error state* after taking action a in state s. The estimated risk expectation is updated similarly to the Q-table values. The state-action value is the weighted mixture of expected accumulated rewards and risks. Learning searches first for a minimum risk policy. After that, an optimal policy below an upper bound for the allowed risk of reaching an error state is sought. (d) Assuming availability of experience tuples from a demonstration of the task performed by an expert [46]. Using state-actions known to be safe from these initial samples, RL is used to achieve a near-optimal performance with respect to the available data. Finally, (e) [47] consider undesirable states as critical, proposing a graph-based algorithm to safely explore a given state-space without reaching a *critical state*. They assume that error states are defined by the magnitude of the rewards, which involves that states *near* critical states can be detected by the rewards. None of these approaches are feasible for over-constrained systems such as L-MCRS for the following reasons: (a) using null state transitions and negative rewards does not encorage the agent to avoid them fast enough because Q-values are weighted sums of the immediate negative rewards and future positive rewards, (b) the system designer requires knowledge of the transition function in order to manually define a state dependent action repertoire, and (c) calculating the risk of reaching an UTS from each state-action pair makes the problem even more complex and requires even more computation.

Modular state-action vetoes. In over-constrained single-agent tasks, we define a partition of the terminal states into two sets $\mathcal{T} = G \cup U$, where $G \subseteq S$ is the set of good termination conditions, and $U \subseteq S$ is the set of UTS. We can also identify the set $T \subseteq S$ of transitory states. We can state that they are characterized by the sign of the achieved rewards as follows:

$$\begin{aligned} G &= \{s \mid s \in S, R(s) > 0\}, \\ T &= \{s \mid s \in S, R(s) = 0\}, \\ U &= \{s \mid s \in U, R(s) < 0\}. \end{aligned}$$

We can decompose further the reward signal as m different signals

$$R(s) = R^G + \sum_{i=1}^{m-1} R_i^U(s),$$

where

- $R^G(s) \geq 0$ is strictly positive only if the task has been successfully accomplished, and zero for transitory states.
- $R_i^U(s) \leq 0, i = 1, \ldots, m-1$ are strictly negative signals only when a certain class of UTS has been reached, e.g. collision, broken physical constraint, etc.

This modular partition into is particularly useful when dealing with over-constrained systems, because it facilitates the learning of how to avoid triggering of the alarm signals R_i^U using only information from the relevant state subspace S_i^U. Under the assumption that the optimal policy will in no case reach a UTS, Safe Modular State-Action Veto (MSAV) policies [8] are useful to boost the exploration efficiency. Let the state space be $S = S_X \times S_i^U$, where S_i^U is the state subspace used by the i-th module. Every time R_i^U is triggered, $|S_X|$ states are effectively vetoed in the complete state space. Thus, MSAV policies will learn $A^e(s)$ faster than any policy learning over the complete state space, and the speed gain introduced is proportional to the number of irrelevant state variables.

The system learns simultaneously the Q-values and the *Safe Action Repertoire* $A^e(s)$

$$A^e(s) = \left\{ a | a \in A \wedge \left(\sum_{s' \in U} P(s, a, s') = 0 \right) \right\},$$

which is the set of actions that cannot lead to a UTS. Vetoes are imposed to risky state-action pairs using only the corresponding state subspace. UTS can be safely avoided and learnt using a Safe-MSAV policy such as

$$\hat{\pi}_{SoftMax}(s, a, \tau) = \begin{cases} 0 & Veto(s, a) \\ \dfrac{\exp^{Q(s,a)/\tau}}{\sum_{a' \in A^e(s)} \exp^{Q(s,a')/\tau}} & \neg Veto(s, a) \end{cases}, \quad (6)$$

where $Veto(s, a)$ is a boolean value that represents whether state-action pair (s, a) should be available to be chosen by the agent or vetoed based on the agent's past experience. Using a Safe-MSAV policy can converge with probability 1 to the optimal values of MDP $< T \cup G, A^e, P, R >$ using standard Q-Learning.

Cooperative Multi-Agent Reinforcement Learning

When the system is composed of several interacting autonomous agents, the Multi-Agent Reinforcement Learning (MARL) problem consists of the search for the joint optimal policies maximizing the reward for each and all agents [9]. In the following, regular characters denote local actions (i.e. a, a_i, A) and corresponding functions (Q, P and R); bold characters denote joint-actions (i.e. $\mathbf{a}$, $\mathbf{a}_i$, $\mathbf{A}$) and corresponding functions. The extension of the MDP for the multi-agent case is a Stochastic Game (SG) defined by tuple $< S, A_1, \ldots, A_N, \mathbf{P}, \mathbf{R}_1, \ldots, \mathbf{R}_N >$, where N is the number of agents, S is the set of environment states, A_i, $i = 1, \ldots, N$ are sets of actions that each agent can execute, yielding the joint action set $\mathbf{A} = A_1 \times \ldots \times A_N$. $\mathbf{P}: S \times \mathbf{A} \times S \rightarrow [0, 1]$ is the probabilistic state transition function $\mathbf{P}(s, \mathbf{a}, s')$ that defines the probability of observing state s' after all agents execute the joint action $\mathbf{a} \in \mathbf{A}$ in state s, and $\mathbf{R}{:}S \times \mathbf{A} \times S \rightarrow \mathbb{R}^N$ are the expected rewards received by agents after transition $(s, \mathbf{a}, s')$.

In the multi-agent case, state transitions are the result of the concurrent joint action $\mathbf{a}_n = [a_{1,n}, \ldots, a_{N,n}]^T \in \mathbf{A}$, $a_{i,k} \in A_i$. As a consequence, the rewards also depend on the collective action. The *local policy* of each agent $\pi_i{:}S \times A_i \rightarrow [0, 1]$ gives the probability of the i^{th} agent executing action a in state s. The local policies are composed into a global *joint policy* $\boldsymbol{\pi}(s, \mathbf{a}) = \{\pi_1(s, a_1), \ldots, \pi_N(s, a_N)\}$.

The SG is fully *cooperative* when the local rewards are identical for all agents $\mathbf{R}_1(s, \mathbf{a}, s') = \mathbf{R}_2(s, \mathbf{a}, s') = \ldots = \mathbf{R}_N(s, \mathbf{a}, s')$,

If a centralized learner exists, the task could be mapped to an MDP, whose centralized Q-function could be expressed as

$$\mathbf{Q}^{\pi}(s, \mathbf{a}) = E^{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, \mathbf{a}_t = \mathbf{a}\right\}, \tag{7}$$

and the optimal policy could be expressed as $\pi^*(s) = \left\{\pi_1^*(s), \ldots, \pi_N^*(s)\right\}$, where

$$\pi_i^*(s) = \arg\max_{a_i} \mathbf{Q}^*(s, [a_1 \ldots a_i \ldots a_N]), \tag{8}$$

where $a_i \in A_i$ and $i = 1, \ldots, N$. When there are more than one optimal action choice, the optimal policy is achieved by a consensus protocol.

Team Q-Learning (*Team-QL*) [24] works on the assumption that there is only one optimal state-action value on each state, so that it updates the local estimates using the standard Q-Learning update rule Eq (5). The Q function depends on the complete joint-action $\mathbf{a} \in \mathbf{A}$, and thus, storage requirements grow with $\mathcal{O}(S \times \mathbf{A})$.

In *Distributed Q-Learning (D-QL)* [23] each agent assumes that the remaining agents will select the optimal action. The virtual centralized state-action Q function Eq (7) is decomposed into smaller local $Q^i(s, a)$ where $a \in A_i$, such that only the maximum value for each local action is stored:

$$Q^i(s, a) = \max_{a_i} \mathbf{Q}(s, [a_1 \ldots a_i \ldots a_N]).$$

This algorithm needs only to be aware of the local action, updating the local Q-values only when the updated state-action value is higher than the previous one:

$$Q^i_t(s, a) \leftarrow \max\left\{Q^i_{t-1}(s, a), r_t + \gamma \max_{a'} Q^i_{t-1}(s', a')\right\}.$$

Because Q-values are only updated when increased, D-QL is expected to estimate the same optimal policies that a centralized learner would learn, without any explicit communication between agents. Its storage requirements grow with $\mathcal{O}(S \times A)$.

More sophisticated approaches include explicit coordination mechanisms. *Coordinated RL* (C-RL) [25] approximates the global joint value function as a linear combination of local value functions [38]. The complexity of agreeing on a globally optimal joint action can be reduced assuming that agents need not coordinate with all remaining agents, but with a smaller subset. These coordination dependencies are defined as a *Coordination-Graph*, denoted $CG(s) = \{V, E\}$, where undirected edges $e_{ij} \in E$ represent a coordination dependence between agents i and j. Each agent has a local Q_i function, which contributes to the global function $Q = \sum_{i=1}^{N} Q_i(s_i, a)$, where $a \in \times A_j$ such that e_{ij} exists in $CG(s)$. This method reduces the state-action space by defining which actions are relevant to each Q_i function.

When implementing the optimal policy, agents need to coordinate the selection of the local actions to build the optimal joint-action. Agents can agree on the optimal joint-action using a *Variable Elimination* (VE) procedure, which maximizes the global value function by maximizing one variable at a time. An agent is chosen to communicate its expected local value for each action to one of its neighbors. Then, this agent can be eliminated from the graph and the selected neighbor can compute the action that maximizes its local value function for the one chosen by the first agent. This procedure is applied to the remaining agents. When only one agent is left, it computes the global maximum and the joint action is propagated with another pass over the CG. This algorithm can be implemented using a simple message-based protocol, and it always computes the global optimal joint-action regardless of the elimination order. However, time constraints can render this approach not suitable for real-time systems.

Cooperative Round-Robin Stochastic Games

To reduce the complexity of dealing with joint actions, we propose to decompose them into a sequence of local actions by considering a Round-Robin schedule assigning turns to execute actions. The Cooperative Round-Robin Stochastic Games formalize this idea.

Definition 1 A *Cooperative Round-Robin Stochastic Game* (C-RR-SG) is a tuple $< S, A_1 \ldots A_N, P, R, \delta >$, where

- N is the number of agents.
- S is the set of states, fully observable by all the agents.
- A_i, $i = 1, \ldots, N$ are the sets of local actions to the i-th agent.
- $P: S \times \cup A_i \times S \rightarrow [0, 1]$, $i = 1, \ldots, N$ is the state transition function $P_t(s, a, s')$ that defines the probability of observing s' after agent $\delta(t)$ executes, at time t, action a from its local action repertoire $A_{\delta(t)}$.
- $R: S \times \cup A_i \times S \rightarrow \mathbb{R}$ is the shared scalar reward signal $R_t(s, a, s')$ received by all agents after executing a local a action from $A_{\delta(t)}$.
- $\delta: \mathbb{R} \rightarrow \{1, \ldots, N\}$ is the turn function implementing the Round-Robin cycle of agent calling for action execution. $\delta(t)$ gives the index of the agent allowed to execute an action at time t (times are relative to the start of the episode). This function is cyclic, $\forall t \in \mathbb{N}; \delta(t) = \delta(t + N)$, therefore, agents are always visired in the same order, so that without loss of generality, we can denote $i + 1$ the agent that will be visited after i.

The state value function $V^{\boldsymbol{\pi}}(s, i)$ gives the value of being in state s for agent $i \in \{1, \ldots, N\}$ under joint policy $\boldsymbol{\pi}$, and it is the expected accumulated discounted rewards following the joint policy $\boldsymbol{\pi}$. The Bellman equation for a joint policy $\boldsymbol{\pi}$ in a C-RR-SG is

$$\begin{aligned} V^{\pi}(s,i) &= E^{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\} \\ &= \sum_{a\in A_i}\pi_i(s,a)\sum_{s'}P(s,a,s') \\ &\cdot \quad [R(s,a,s') + \gamma V^{\pi}(s', i+1)], \end{aligned}$$

where $E^{\boldsymbol{\pi}}$ represents the expectation from time t onwards, following joint policy $\boldsymbol{\pi}$. The optimal state value $V^*(s, i)$ is given by:

$$V^*(s,i) = \max_{a \in A_i} \left\{ \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma V^*(s', i+1)] \right\}. \tag{9}$$

The state-action value function for agent i following joint policy $\boldsymbol{\pi}$ can be expressed as

$$Q^{\pi}(s,a,i) = \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s', i+1)] \tag{10}$$

and the optimal state-action value function for each agent i is

$$Q^*(s,a,i) = \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma \max_{a' \in A_j} Q^*(s', a', i+1)].$$

Centralized Q-Learning for C-RR-SG

The straightforward approach to learn the optimal policy for a C-RR-SG is applying the single-agent Q-Learning update:

$$\begin{aligned} Q_t(s,a,\delta(t)) \quad \leftarrow \quad & (1-\alpha_t) Q_{t-1}(s,a,\delta(t-1)) \\ & + \alpha_t [r_t + \gamma \max_{a'} Q_{t-1}(s_{t+1}, a', \delta(t-1))]. \end{aligned} \tag{11}$$

Proposition 1 *A centralized single-agent Q-Learning training of a C-RR-SG by the rule of Eq (11) will converge to* Q*(s, a, i) *if each agent fulfills the conditions for convergence of the single agent Q-learning.*

Proof. If we are able to map the C-RR-SG into an MDP, then the proof of the proposition follows from the convergence of single agent Q-learning. The map is as follows: The MDP state space S is the same of the C-RR-SG, since it is defined by a set of global variables visible by all agents. The MDP set of actions is the union of the local sets of actions $A = \bigcup_{i=1}^{N} A_i$, where actions lose the direct identification with the agent. The MDP state transition probability is built by considering $P(s, a, s') = P(s, a_{\delta(t)}, s')$, that is, actions are sequenced according to the turn function. The MDP reward function is built by $R(s') = R(s, a_{\delta(t)}, s')$, that is, we record the reward at the arriving state. The C-RR-SG turn function removes the uncertainty and concurrency about the joint agent actions, therefore the MDP policy can be stated as $\pi(s_t) = \boldsymbol{\pi}_{\delta(t)}(s_t)$, that is, we apply at each time the local policy of the agent selected by the turn function. The MDP Q-table is built by collapsing all the agent Q-tables into a monolithic one by applying the turn function, i.e. $Q_t(s, a) = Q(s, a, \delta(t))$. Single-agent Q-Learning converges to optimal $Q^*(s, a)$ values under two conditions [5]: all state-action pairs (s, a), $a \in \cup A_i$, $s \in S$ may be visited infinite times, and the learning gain α_t is decreased according to the conditions for convergence of the stochastic gradient, i.e. $\sum_t \alpha_t = \infty$ and $\sum_t (\alpha_t)^2 < \infty$. The second condition is easy to fulfill, by selecting an appropriate schedule for α_t. To prove the first condition, it is enough to show that the MDP obtained from the C-RR-SG is an irreducible Markov chain. This follows inmediately if the sate transition matrix P does not contain zero elements, and the action selection policy attributes nonzero probability to all actions in every state. The latter is assured by the use of exploratory policies (i.e. SoftMax), the former is ensured by the fact that each individual agent fullfills the conditions for convergence, i.e. $P(s, a_{\delta(t)}, s') > 0$ therefore the MDP state transition probabilities will be positive.

This centralized learning process represents the straightforward tabular implementation of centralized Q-Learning on a C-RR-SG. This algorithm is able to learn a set of locally optimal policies which result in a globally optimal policy $\boldsymbol{\pi}^*$. This centralized implementation, while conceptually simple, has a big drawback: it assumes an omniscient centralized learner. This is hardly true in real-world applications because of the communication requirements, especially in an MCRS scenario, where noisy and faulty communications make this problem even worse. From the point of view of computational complexity, this centralized learner should store $|S \times \cup A_i|$, $i = 1, \ldots, N$ entries in its Q-table.

Distributed Round-Robin Q-Learning

In distributed implementations the state-action value matrix $Q(s, a, i)$ is decomposed into local independent matrices $Q^i(s, a)$ corresponding to the agent owning turn i, so that all information is distributed $Q(s, a, i) = Q^i(s, a)$. We present two different distributed Round-Robin Q-Learning algorithms (D-RR-QL) update rules for stochastic games of type C-RR-SG that differ in their communication strategies: the first one requires at each time-step to send information from the current agent to the next in turn according to the Round-Robin schedule, the second is communication-free. In both cases, we prove convergence to an optimal policy.

In the first D-RR-QL update rule, the Q-table is updated using the information from the next agent in the RR scheduling cycle, so that local Q-table updating does not need to wait the full cycle. However it requires that agent $i + 1$ informs agent i of its optimal policy given by $\max_{a'} Q^j_{t-1}(s_{t+1}, a')$. Once this value is communicated, the i-th agent can update its own Q-table using the following update rule:

$$\begin{aligned} Q^i_t(s,a) \quad \leftarrow \quad & (1-\alpha_t)Q^i_{t-1}(s,a) \\ & + \alpha_t[r_t + \gamma \max_{a'} Q^j_{t-1}(s_{t+1}, a')]. \end{aligned}$$

The D-RR-QL algorithm using this rule inherits its convergence properties from the centralized single-agent algorithm, and needs not additional proof.

The communication-free D-RR-QL algorithm is specified in Algorithm 1. In this algorithm the local Q^i table is updated at the end of an RR cycle using the information of the rewards that have been broadcasted to all agents because we are dealing with a fully cooperative MARL, according to the following rule:

$$\begin{aligned} Q^i_t(s,a) \quad = \quad & (1-\alpha_t)Q^i_{t-N}(s,a) \\ & + \alpha_t\left[\sum_{k=0}^{N-1}\gamma^k r_{t+k} + \gamma^N \max_{a'} Q^i_t(s_{t+N}, a')\right] \end{aligned} \tag{12}$$

applied when $s_t = s$, $a_t = a$, $\delta(t) = \delta(t - N) = i$. This update rule allows each agent to update its local Q^i-table without the need to know the Q-tables of other agents. This is the communication-free implementation of the D-RR-QL algorithm which will be the subject of our experiments.

Algorithm 1 Algorithm executed by agent i for the estimation of the local Q^i table according to the communication-free D-RR-QL algorithm assuming a global time counter visible to all agents.

```
Initialize $\left[Q^i_0(s,a,i) = 0; s \in S; a \in A; i = 1 \ldots N\right]$ arbitrarily
Repeat (for each episode) $n$:
  Repeat (for each step $t$ of episode):
  • Wait until $\delta(t) = i$
  • Observe current state $s_t$
  • Select and execute action $a_t$
  • Give turn to next agent in RR
  • Observe the rewards $r_t$, ..., $r_{t+N-?1}$ of a complete RR cycle and new
    state $s_{t+N}$ after the RR cycle.
  • Adaptation of the local $Q^i$ table
    – if $s_t = s$, $a_t = a$, $i = \delta(t)$
```

$$
\begin{aligned}
Q_t^i(s,a) &= (1-\alpha_t)Q_{t-1}^i(s,a) \\
&+ \alpha_t\left[\sum_{k=0}^{N-1}\gamma^k r_{t+k} + \gamma^N \max_{a'} Q_t^i(s_{t+N},a')\right]
\end{aligned}
$$

```
  – $Q_t^i(s,a) = Q_{t-1}^i(s,a)$ otherwise
until $s_{t+?1}$ is terminal
```

Proposition 2 *Convergence of the communication-free D-RR-QL of Algorithm 1 to the optimal policy, $Q_t^i(s,a) \to Q^*(s,a,i)$ as* t $\to \infty$, *for a given a C-RR-SG* $\langle$S, A$_1$. . .A$_N$, P, R, δ$\rangle$ *is guaranteed when each agent fulfills the conditions of convergence of single-agent Q-Learning in a MDP.*

Proof. Let us consider for agent *i* the state value after N steps, combining the accumulation of the sequence of *N* rewards observed during an RR cycle starting and ending in agent *i* and the forgetting factor, which is given by

$$R_t^{(N)} = \sum_{k=0}^{N-1}\gamma^k r_{t+k+1} + \gamma^N V_t(s_{t+N}, i).$$

For agent *i*, the *corrected N-step truncated reward return* [3], which is is the perceived increment of the state value after N update steps of the learning process is given by:

$$\Delta V_t(s_t, i) = \alpha[R_t^{(N)} - V_t(s_t, i)], \tag{13}$$

where $V_t(s_t, \delta(t))$ is the value of being in state s_t at time t. For single agent learning, N-step TD Methods converge to optimal value functions $V^*(s)$ due to their *error reduction property* [4]:

$$\max_s |E^\pi\left\{R_t^{(N)} \mid s_t = s\right\} - V^*(s)| \le \gamma^N \max_s |V_t(s_t) - V^*(s)|,$$

which means that the error commited by worst one-step ahead estimate $V_t(s_t)$ is an upper bound of the error committed by the N-step ahead estimate $E^{\text{p}}\left\{R_t^{(N)} \mid s_t = s\right\}$ of the optimal state value function. Besides, this error bound can be made as tight as desired because the factor $\gamma^N \to 0$ as $N \to \infty$. From Eq (13) we can derive the update rule for the state-action table of Eq (12) used by Algorithm 1:

$$
\begin{aligned}
\Delta Q_t^i(s_t,a) &= \alpha\left[\sum_{k=0}^{N-1}\gamma^k r_{t+k+1} + \gamma^N V_t(s_{t+N}, i)\right] \\
&= \alpha\left[\sum_{k=0}^{N-1}\gamma^k r_{t+k+1} + \gamma^N \max_{a'} Q_t^i(s_{t+N}, a')\right].
\end{aligned}
$$

Therefore, in the communication-free D-RR-QL of Algorithm 1 each agent performs a N-step ahead update of its local state-action value function, where the accumulated rewards are exogenous variables to the agent. The local updating of $Q^i(s, a)$ are time-interleaved Q-learning processes, which converge to the optimal $Q^*(s, a, i)$ if the stochastic gradient conditions of [5] hold for each agent, i.e. it can visit all state-action pairs infinitely often. Each agent computations are separated from the others, though they share the same global state and rewards so they can not be considered statistically independent. Statistical independence is not a requirement for convergence of the computationally independent learning processes.

Composition of Concurrent Joint-Action Policies

A C-RR-SG is an approximation to the original Cooperative Stochastic Game (C-SG) problem, because agents are forced to carry out actions following a predefined sequential order. Policies learned using this approximation are very likely to be suboptimal in the original C-SG environment, because the discounted reward approach assumes that the task completion instant is critical for the definition of optimal policies. The problem posed in this section is how to compose the local policies learned by the separate agents using D-RR-QL into the optimal policy for the original C-SG, where actions can be executed concurrently. Therefore, we deal with a combinatorial optimization problem. In this section, we present a sequential greedy coordination algorithm to build an approximation of the C-SG optimal policy $\boldsymbol{\pi}(s,\mathbf{a})$ from the distributed Q-tables learned using D-RR-QL.

Definition 2 A *C-RR-SG* $< S, A_1 \ldots A_N, P, R, \delta >$ is a *sequential realization* of a *C-SG* $< S, A_1 \ldots A_N, \mathbf{P}, \mathbf{R} >$ if the following two properties hold:

- $\forall s, s', \mathbf{a}, i; \mathbf{P}(s, \mathbf{a}_i, s') = P(s, a_i, s')$
- $\forall s, s', \mathbf{a}, i; \mathbf{R}(s, \mathbf{a}_i, s') = R(s, a_i, s')$,
  where joint-action $\mathbf{a}_i$ is defined as the joint-action in which only the i-th agent performs an action, i.e. $\mathbf{a}_i = [\emptyset, \ldots, \emptyset, a_i, \emptyset, \ldots, \emptyset]$. Null action Ø is defined as the action that does not perform any state transition: $\forall s; P(s, \emptyset, s) = 1$.

The transition probabilities and *discounted* rewards of a joint-action $\mathbf{a} = [a_1, a_2, \ldots, a_N]$, $a_i \in A_i$, are approximated by the sequential execution of agents' actions following RR scheduling δ:

- $\mathbf{P}(s, \mathbf{a}, s') \simeq P(s, a_{\delta(1)}, a_{\delta(2)} \ldots a_{\delta(N)}, s')$
- $\mathbf{R}(s, \mathbf{a}, s') \simeq R(s, a_{\delta(1)}, a_{\delta(2)} \ldots a_{\delta(N)}, s')$,

where $P(s, a_{\delta(1)}, a_{\delta(2)} \ldots a_{\delta(N)}, s')$ and $R(s, a_{\delta(1)}, a_{\delta(2)} \ldots a_{\delta(N)}, s')$ denote the probability of reaching state s' after executing actions specified in $\mathbf{a}$ in the sequence established by δ, and the expected reward of this transition, respectively. Under this assumption, Q-values for joint-actions can be approximated by composition of the transition and reward functions of local actions. The state-action value for composed joint-actions can be defined as

$$\begin{aligned} \mathbf{Q}^{\pi}(s_0, \mathbf{a}) &= \sum_{s_1 \ldots s_N \in S} \left(\prod_{i=0}^{N-1} P(s_i, a_{i+1}, s_{i+1}) \right) \\ &\cdot \left\{ \sum_{i=0}^{N-1} \gamma^i \cdot R(s_i, a_{i+1}, s_{i+1}) + \gamma^N \cdot \max_{a'} Q^{\pi}(s_N, a', \delta(1)) \right\} \end{aligned} \tag{14}$$

where $\mathbf{a} = [a_1\, a_2 \ldots a_N]$, $a_i \in A_{\delta(i)}$. Note that rewards inside a joint-action are weighted as in the C-RR-SG.

Greedy selection of optimal joint policy The combinatorial optimization problem of chosing the best ordering of the local actions has been tackled as a greedy variable selection process [25] requiring agents to store estimations of the transition probabilities $\hat{P}_i(s, a, s')$ and rewards $\hat{R}_i(s)$ for local actions $a \in A_i$. Observed rewards can be stored with minimal storage requirements. Typical delayed reward scenarios involve a small number of terminal states that receive a non-null reward. Moreover, MSAV policies store state-action pairs with negative rewards using only the relevant state variables. Therefore, an agent can keep track of those states that have yielded an immediate positive reward in the past as a set of pairs $S_i^r = \{\langle s_i, r_i \rangle ; i = 1, \ldots, k\}$, where k is the number of elements in the set. Every time a reward $R(s, a, s') > 0$ is observed, the pair $\langle s', R(s, a, s') \rangle$ is added to the set S_i^r, if it wasn't already in it. The reward function estimate can be written as

$$\hat{R}_i(s) = \begin{cases} r_i & \mathit{if}\ \exists i; 1 \leq i \leq k \wedge s_i = s \\ 0 & \mathit{otherwise}, \end{cases},$$

where s_i represents the i^{th} state in set S_r. At this point, we are assuming that rewards are deterministic functions of the state where they are received.

The greedy maximization algorithm to obtain the optimal joint policy maximizing the state-value function of Eq (14) follows a message-passing scheme. Each agent selects its optimal action according to the local Q-table, then the agent forwards to the next agent in the RR schedule the list of positive probability state transitions after executing the selected action. Let the system start in a state s_0, agent $\delta(1)$ selects $a_1 = \arg\max_a \{Q^{\delta(1)}(s_0, a)\}$ and sends a message $\langle s_1^j, \hat{P}_{\delta(1)}(s_0, a_1, s_1^j); j = 1, \ldots, k_1 \rangle$ to agent $\delta(2)$, where k_1 is the number of states s_1^j such that $\hat{P}_{\delta(1)}(s_0, a_1, s_1^i) > 0$. Then, agent $\delta(2)$ selects its optimal action on the basis of the possible states:

$$a_2 = \arg\max_a \left\{ \sum_{j=1 \ldots k_1} \hat{P}_{\delta(1)}(s_0, a_1, s_1^j) \cdot (\hat{R}_{\delta(1)}(s_1^i) + \gamma^2 \cdot Q^{\delta(2)}(s_1^j, a)) \right\},$$

and sends a message $\langle s_2^j, \hat{P}(s_0, a_1, , a_2 s_2^j); j = 1, \ldots, k_2 \rangle$ to the next agent $\delta(3)$, where

$$\hat{P}(s_0, a_1, a_2, s_2^j) = \sum_{j=1 \ldots k_1} \hat{P}_{\delta(1)}(s_0, a_1, s_1^j) \cdot \hat{P}_{\delta(2)}(s_1^j, a_2, s_2^j),$$

and k_2 is the number of states s_2^i such that $\hat{P}(s_0, a_1, a_2, s_2^i) > 0$.

Generalizing this process, agent $\delta(i)$ receives a message $\langle s_{i-1}^j, \hat{P}(s_0, a_1 \ldots a_{i-1}, s_{i-1}^j); j = 1, \ldots, k_{i-1} \rangle$, then selects its local optimal action

$$a_i = \arg\max_a \left\{ \sum_{j=1 \ldots k_{i-1}} \hat{P}(s_0, a_1 \ldots a_{i-1}, s_{i-1}^j) \cdot (\hat{R}_{\delta(i-1)}(s_{i-1}^j) + \gamma^i \cdot Q^{\delta(i)}(s_{i-1}^j, a)), \right.$$

and, finally, sends message $\langle s_i^j, \hat{P}(s_0, a_1 \ldots a_i, s_i^j); j = 1, \ldots, k_i \rangle$ to agent $\delta(i + 1)$. When the procedure reaches the last agent, all agents execute a joint-action $\mathbf{a} = [a_1, a_2, \ldots, a_N]$ with is the greedy solution to the following minimization problem:

$$\pi_D(s) = \arg\max_{a_1, a_2 \ldots a_N} \left\{ \sum_{j=1 \ldots k_N} \hat{P}(s_0, a_1 \ldots a_N, s_N^j) \cdot (\hat{R}_{\delta(N)}(s_N^j) + \gamma^N \max_{a'} Q^{\delta(N)}(s_N^j, a')) \right\}. \quad (15)$$

For completeness, terminal states must be taken care of, because we are composing cycles of actions. If a terminal state is reached, there will not exist transition probability information from this state onwards. We can address this potential implementation issue by leaving unaltered the entries in the incoming messages for which no transition probability is available, so that the remaining agents will perform no action at all.

Proposition 3 *The action selection policy defined by Eq (15) approximates the optimal greedy joint-policy given by Eq (8).*

Proof. The transition and reward functions are approximated by $\mathbf{P}(s, \mathbf{a}, s') \simeq \hat{P}\left(s, a_{\delta(1)}, a_{\delta(2)} \ldots a_{\delta(N)}, s'\right)$ and $\mathbf{R}(s, \mathbf{a}, s') \simeq \hat{R}\left(s, a_{\delta(1)}, a_{\delta(2)} \ldots a_{\delta(N)}, s'\right)$, therefore, we can aggreagate local actions into a joint action, substituting both approximated terms in Eq (15) rewriting it as follows:

$$\pi_D(s) \quad \simeq \arg \quad \max_{\mathbf{a}} \left\{ \mathbf{P}(s, \mathbf{a}, s') \cdot (\mathbf{R}(s, \mathbf{a}, s') + \gamma^N \max_{a'} Q^{\delta(N)}(s', a')) \right\}.$$

Furthermore, $\text{argmax}_{a'}\ Q^{\delta(N)}(s', a') = \text{argmax}_{\mathbf{a}'}\ \mathbf{Q}^{\mathrm{P}}(s', a')$, we conclude that the joint policy approximates the optimal joint policy learned by a centralized learner given by Eq (8): $\boldsymbol{\pi}_D(s) \simeq \boldsymbol{\pi}^*(s)$.

This approximation has been tested experimentally in environments with delayed rewards, with positive results. Unlike the technique presented in [25], where the message network could have different topologies, our message passing process only needs one cycle to finish. If our assumptions are completely fulfilled, the system is able to learn an optimal policy for $< T \cup G, \mathbf{A}^e, \mathbf{P}, \mathbf{R} >$.

D-RR-QL with Modular State-Action Vetoes

An additional advantage of D-RR-QL is that it allows us to use MSAV in a multi-agent environments. Since we assume that a joint-action is approximately the same as the equivalent sequence of local actions, agents can veto actions on their local action space. Agents take actions one by one during the cycle of execution and thus have no interference from the rest of agents. Of course, this requires taking N time-steps to perform N local actions instead of a single joint-action, and the execution time will grow linearly to the number of agents. Nevertheless, this is neglectable in the case of overconstrained environments, where learning in an effective manner $A^e(s)$ is critical.

The speed boost introduced by the use of MSAV in such multi-agent environments is even bigger than in the single-agent case (Section *Modular State-Action Vetoes*). Let the state be defined $S = S_X \times S_i^U$, being S_i^U the state subspace used by the i-th *Veto-Module*, and let the joint-action space be defined $\mathbf{A} = A_j \times A^-$, where A_j is j-th agent's local action space, and $A^- = A_1 \times A_{j-1} \times A_{j+1} \times \ldots \times A_N$. Whenever an agent vetoes a state-action pair in its own state-action subspace, it is effectively vetoing $|S_X \times A^-|$ state-action pairs in the joint state-action space. The multi-agent Safe-MSAV architecture can be expected to avoid UTS much faster than any other exploration strategy operating on the complete state space S.

Experiments

Linked Multi-Component Robotic Systems (L-MCRS) are over-constrained systems which can benefit from the use of D-RR-QL with MSAV. The hose transportation task is a paradigmatic application of L-MCRS: A set of $N = 4$ robots is attached to a hose at fixed points and the hose is modeled as a segment line lying between robots (Fig 3). This simplification of the original environment in which GEDS are used to model the hose is used as a training arena where agents learn the basic skills, and then they better tune the policies learnt in the GEDS environment. The goal is to transport the tip of the hose to a predefined goal position (the reward received is 10), while the other extreme is attached to a fixed position, a source, arbitrarily set at the center of the working space. The simulated world consists of 19×19 cells and each robot is controlled by an agent which can execute five actions: $A_i = \{up_i, down_i, left_i, right_i, none_i\}$. The four constraints considered are the ones enumerated in Section *Problem Statement*, and whenever one of these termination conditions is met, the system generates a negative reward (−1) and the environment is reset to the initial setting (Fig 3). The system state is defined by the following state variables:

- The relative positions of the *N* robots. Relative positions are calculated with respect to the previous robot in the formation (source position (0,0) for the first robot).
- Self-position in absolute coordinates. Each robot perceives its own absolute position.
- Four flag variables indicating whether there is an obstacle (hose, robot) in each of the four possible directions a robot can move.

Benchmark algorithms (D-QL, Team-QL and Coordinated-RL) only use robot relative positions because the remaining state variables are a linear function of them and these algorithms cannot benefit from the additional information provided by the extended set of variables. For instance, it is of no use to include the obstacle flag variables because they are a function of the relative positions of the robots and thus, can only take one value for any given set of relative positions. In order to best learn how to avoid breaking a physical constraint corresponding to an UTS, we used three Veto-modules within each agent's MSAV policiy, each of which in charge of a specific physical constraint:

- Overstretching the hose. This module receives a negative reward every time a hose segment is overstretched, and its local state space encompasses the relative positions of the robots preceding and following the robot which executed the action that led to over-stretching its hose segment.
- Collisions. Every time a robot takes an action that makes it collide with any other, this module receives a negative reward. The local state space is composed by the obstacle flag variables.
- Simulation bounds. This last module uses only its robot's absolute position and receives a negative reward every time a robot gets out of the simulation grid.

Each of the four generated negative rewards was fed to a specific module learning the veto of specific undesired state-actions using the value of the *relevant state variables* [8].

Because the amount of episodes and the degree of exploration is key to this particular kind of tasks, experiments were run with different number of episodes n_e. D-RR-QL with MSAV was run using the Safe Soft-Max policy Eq (6), while the rest of algorithms used regular Soft-Max exploration Eq (4). We also tested $\epsilon - greedy$ policies but the experiments yielded slightly worse results and thus, will not be reported here. Initially, the temperature parameter was set $\tau = 10$ and decreased at the end of each episode with $\triangle\tau = -10/n_e$. Starting from episode number 500, the learnt policy was evaluated using greedy action selection each 500 episodes. This is the reason why data plots start from episode 500 instead of episode 0. Each experiment was run 3 times and the results were averaged. The performance was measured using two different criteria: (a) the accumulated discounted rewards, which is the typical performance measurement in the RL field, and (b) the number of cells reached with the tip of the hose, which allows us to intuitively assess how effective the exploration of the state-space is.

We implemented ourselves these experiments in C/C++. The source code used in our experiments and can be accesed publicly at: http://www.ehu.es/ccwintco/index.php?title=D-RR-QL.

Experiment 1

In this first experiment, we tested two different values of $n_e = \{10^4, 2 \cdot 10^4\}$. Figs 4 and 5 show the accumulated rewards obtained by agents in the two different settings. The learning gain by

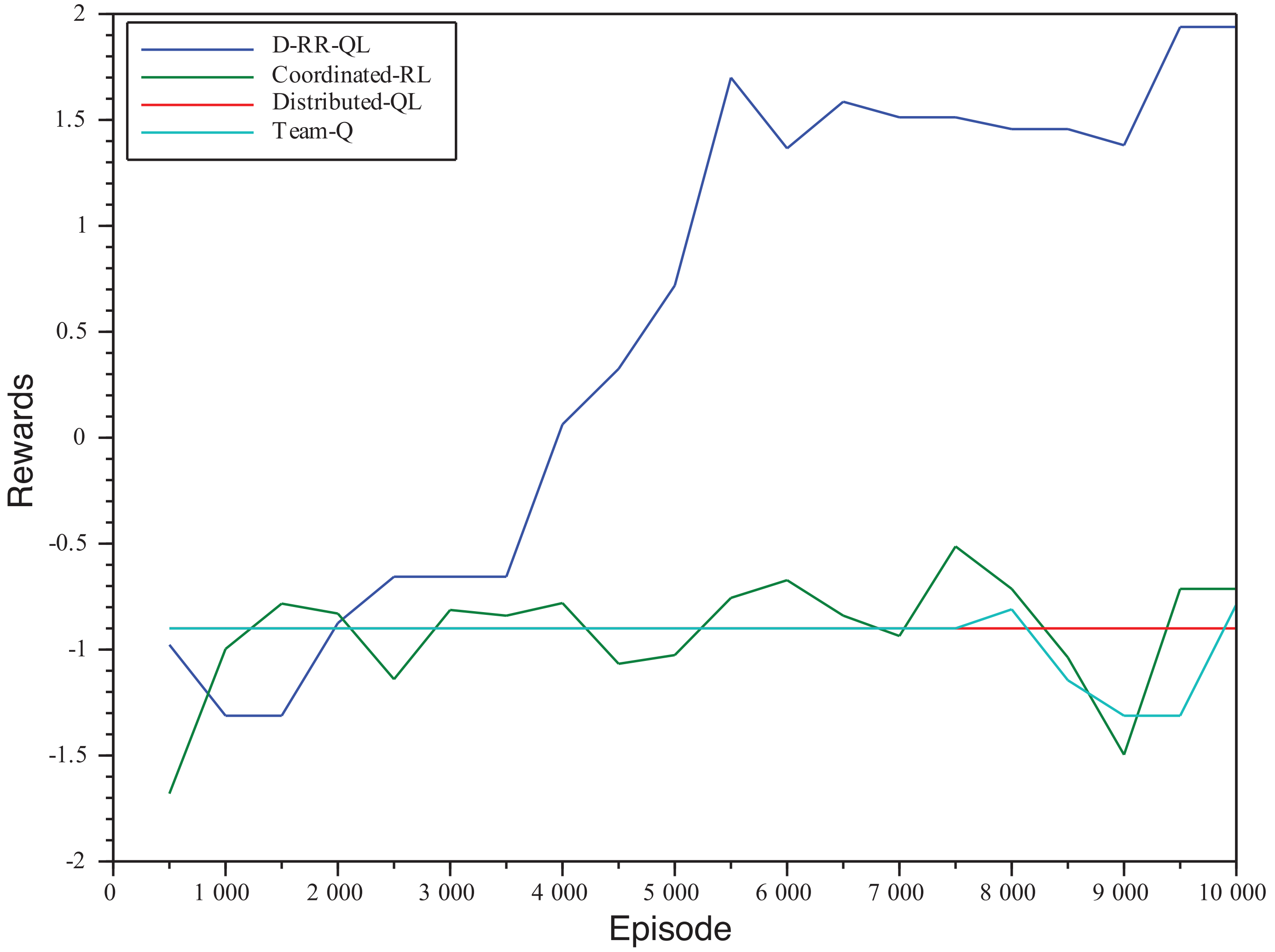

Fig 4. Rewards obtained by the learnt greedy policies with $n_e = 10^4$.

doi:10.1371/journal.pone.0127129.g004

using D-RR-QL with MSAV is huge. In both cases, it is the only algorithm able to obtain a positive accumulated discounted reward in the evaluation episodes. It only requires about 4,000 episodes for our method to learn a greedy policy that can reach the goal position. The rest of algorithms are not only incapable of avoiding UTS, but also of *even reaching the goal cell a single time*.

Figs 6 and 7 on the other hand show the number of different states reached with the tip of the hose. The plot clearly shows that the exploration of the state-space is much more efficient when using MSAV than by means of standard Soft-Max policies. By the time D-RR-QL has been able to learn a greedy policy able to reach the goal, the system has already reached about 275 different cells with the tip of the hose. On the other hand, none of the other algorithms is able to reach more than 70 different cells throughout the complete experiment.

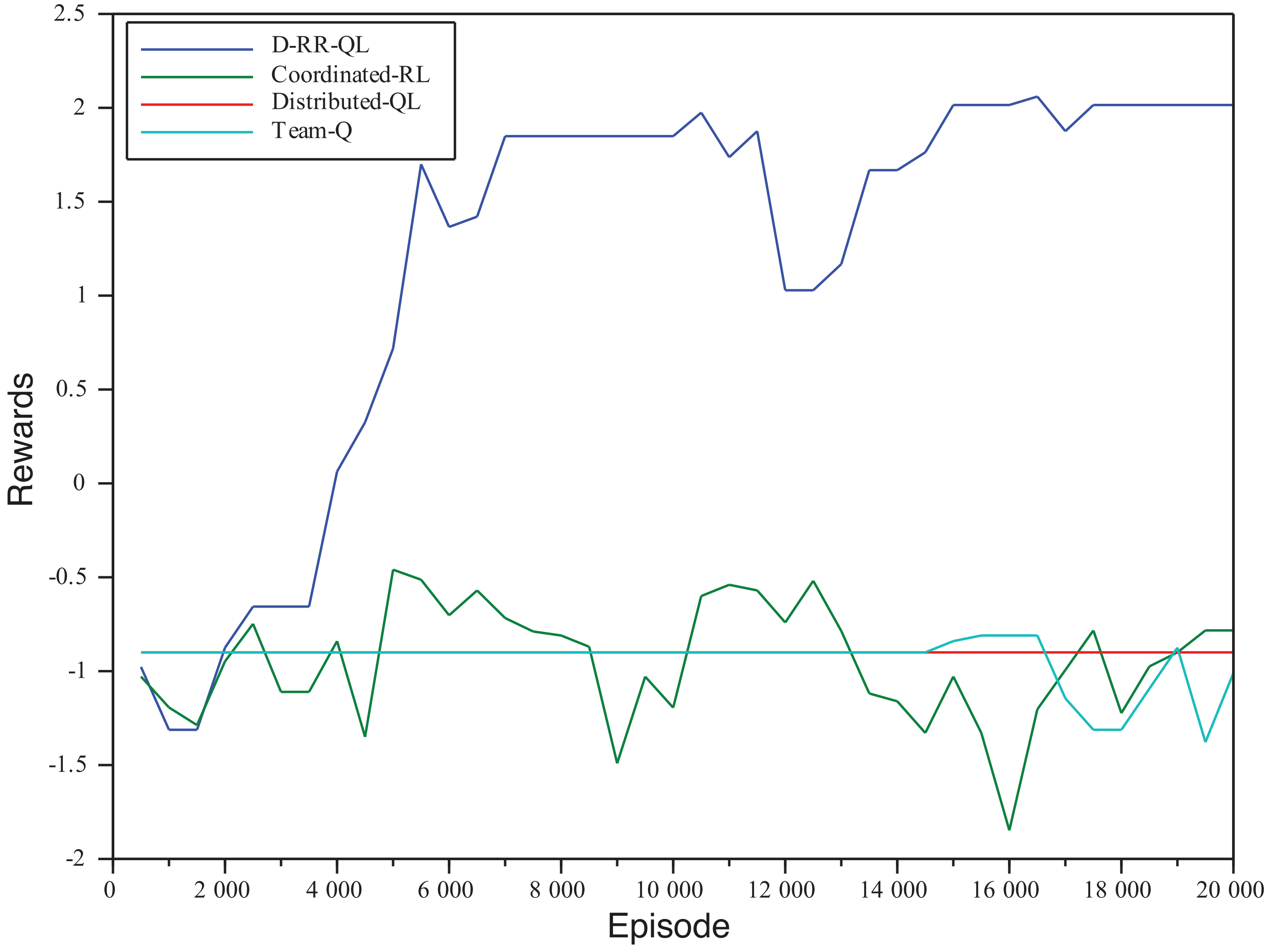

Fig 5. Rewards obtained by the learnt greedy policies with $n_e = 2 \cdot 10^4$.

doi:10.1371/journal.pone.0127129.g005

Experiment 2

In order to get a clearer idea of how far the benchmark algorithms are from the goal of learning a greedy policy that reaches the goal position, we ran a second set of experiments where agents were allowed much more time for exploration of the environment ($n_e = 5 \cdot 10^5$ and $n_e = 10^6$). We only report the number of reached cells with the tip of the hose, because the accumulated discounted rewards are similar to those obtained in the previous experiment, and offer no further information. Both Soft-Max and $\epsilon - greedy$ policies were tested and, in this case, slightly better results were obtained by using $\epsilon - greedy$. Initially, $\epsilon = 1$ and every episode it was decreased with $\triangle\epsilon = -1/n_e$. This is paradoxical considering the better results obtained by Soft-Max policies in the previous experiment. Our educated guess is that this is due to using the same decay schedule for the two exploration parameters ϵ and τ. While they have a similar function, ϵ is the probability of exploring each time-step and τ shapes the probability

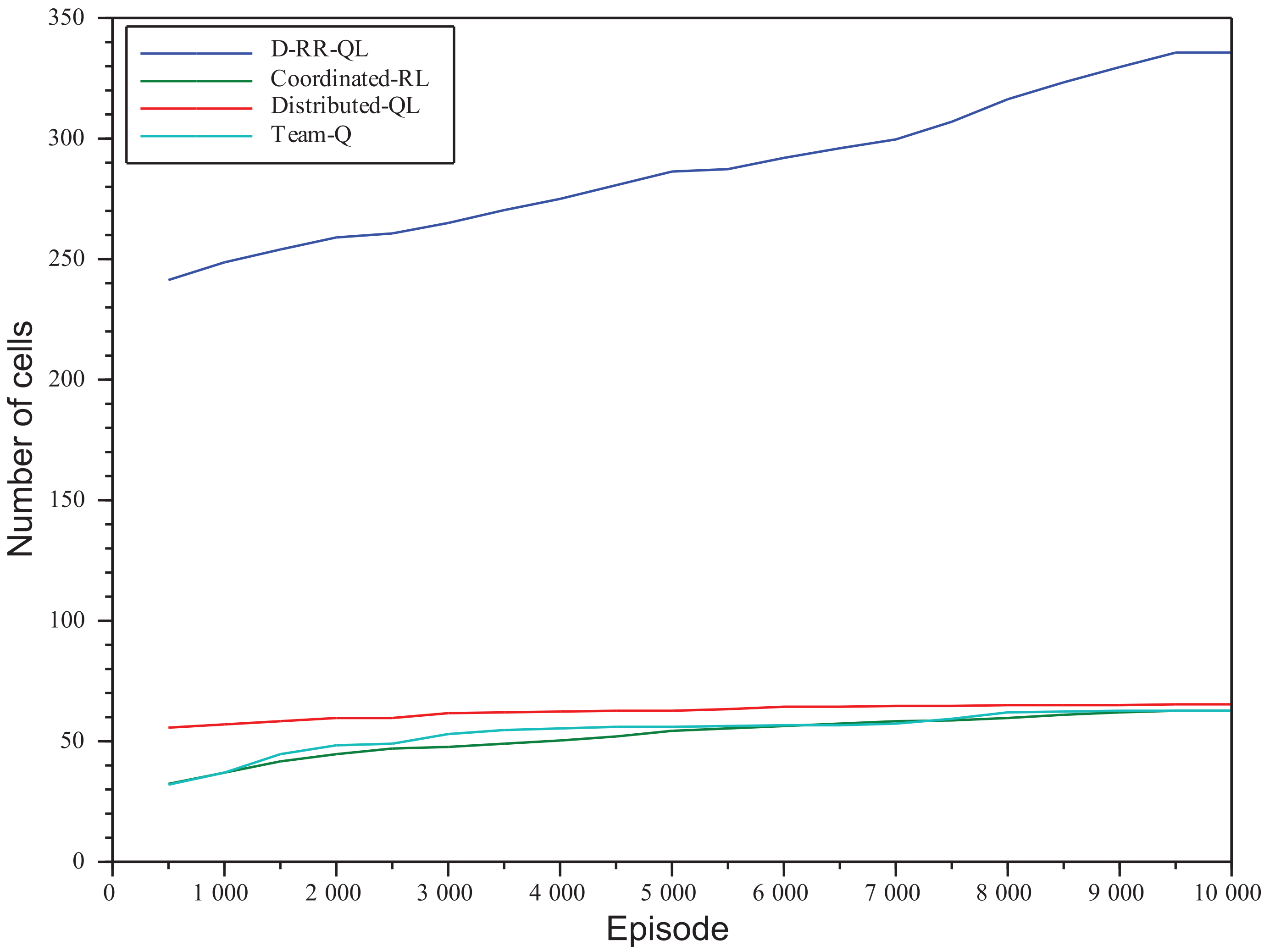

Fig 6. Number of cells reached with the tip of the hose with $n_e = 10^4$, separate plots when applying D-RR-QL, Distributed-QL, Coordinated-RL, and Team-QL algorithms.

doi:10.1371/journal.pone.0127129.g006

distribution of the action selection algorithm. Moreover, as the Soft-Max algorithm depends on the magnitude on the actual estimations of the value function, the efficiency of the exploration algorithm needs not improve linearly to the number of episodes allowed to the agents.

Figs 8 and 9 plot the number of cells and, although the maximum amount of episodes is 2 orders or magnitude greater than in the previous experiments, the system is only able to reach about 120 cells with the mobile tip of the hose in the case of Coordinated-RL, which outperforms the other two algorithms. Next comes Distributed-QL with 92 cells visited, and finally, Team-QL is only able to reach 74 cells. It is very important to notice that, once again, *neither of these algorithms was able to reach the goal cell even once* during the experiments. For the sake of comparison with the D-RR-QL and MSAV approach, let us recall that the algorithm was first able to reach the goal greedily selecting the best action when 275 cells had been visited with the tip of the hose. This is a clear indication of how inefficient the exploration of over-constrained environments by standard action selection policies is: if we take the number of cells

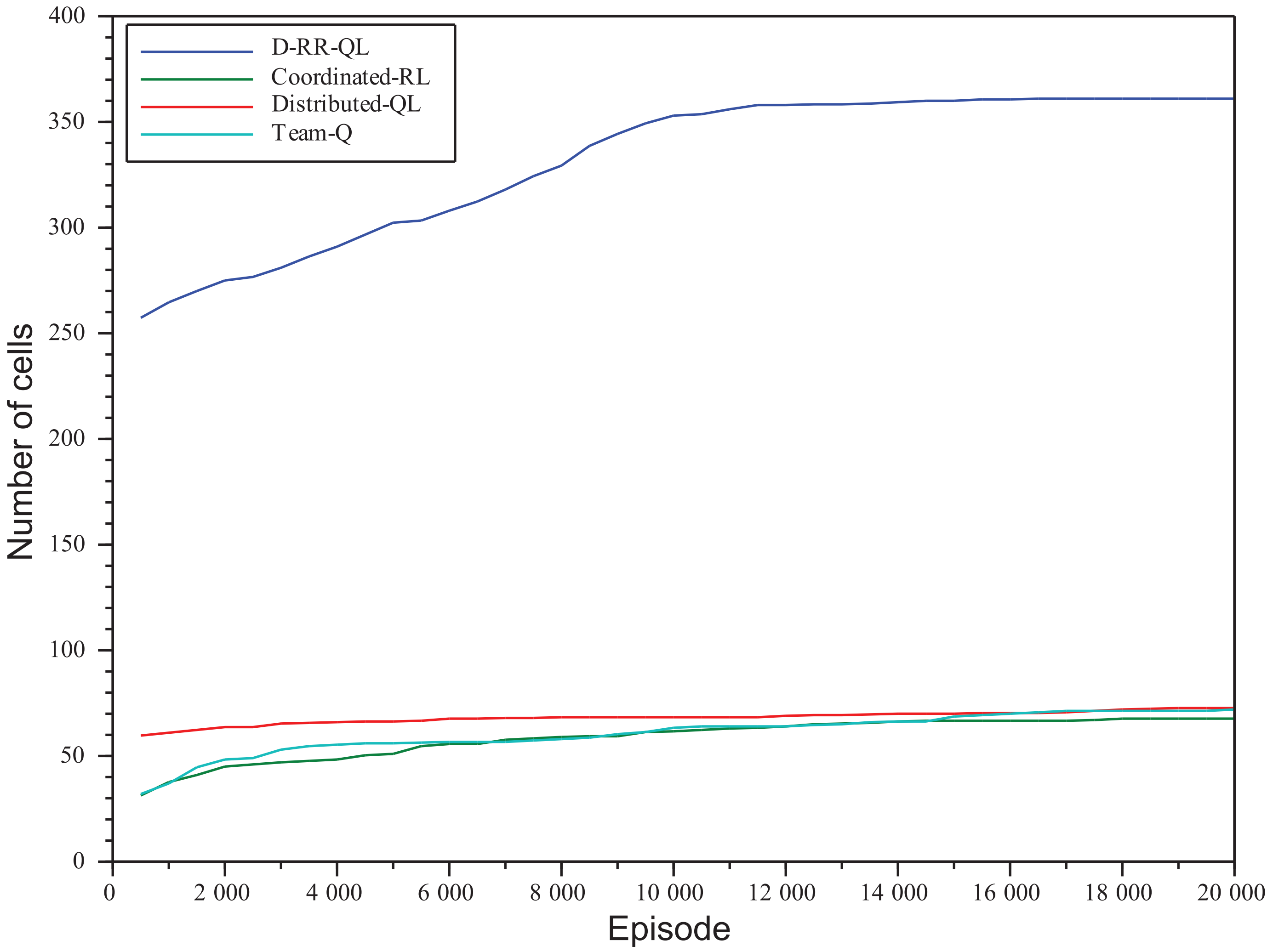

Fig 7. Number of cells reached with the tip of the hose with $n_e = 2 \cdot 10^4$, separate plots when applying D-RR-QL, Distributed-QL, Coordinated-RL, and Team-QL algorithms.

doi:10.1371/journal.pone.0127129.g007

visited by D-RR-QL before it can learn to reach the goal cell as a reference of how much exploration is needed before a goal state has been discovered, all the competing algorithms are very far from this number even if allowed to explore two orders of magnitude more.

Conclusions

This paper formalizes and gives convergence proof of a distributed Q-Learning algorithm for MARL problems amenable to embed Modular State-Action Veto (MSAV) policies which improve performance in over-constrained environments. First, we have presented agent interaction model as a Cooperative Round Robin Stochastic Game (C-RR-SG), in which agents select and take actions following some predefined order. A communication-free distributed implementation called Distributed Round Robin Q-learning (D-RR-QL) has been proposed, providing a proof of its convergence to the optimal policy on a C-RR-SG. We have also presented a message-based procedure to obtain the optimal policy for the original cooperative

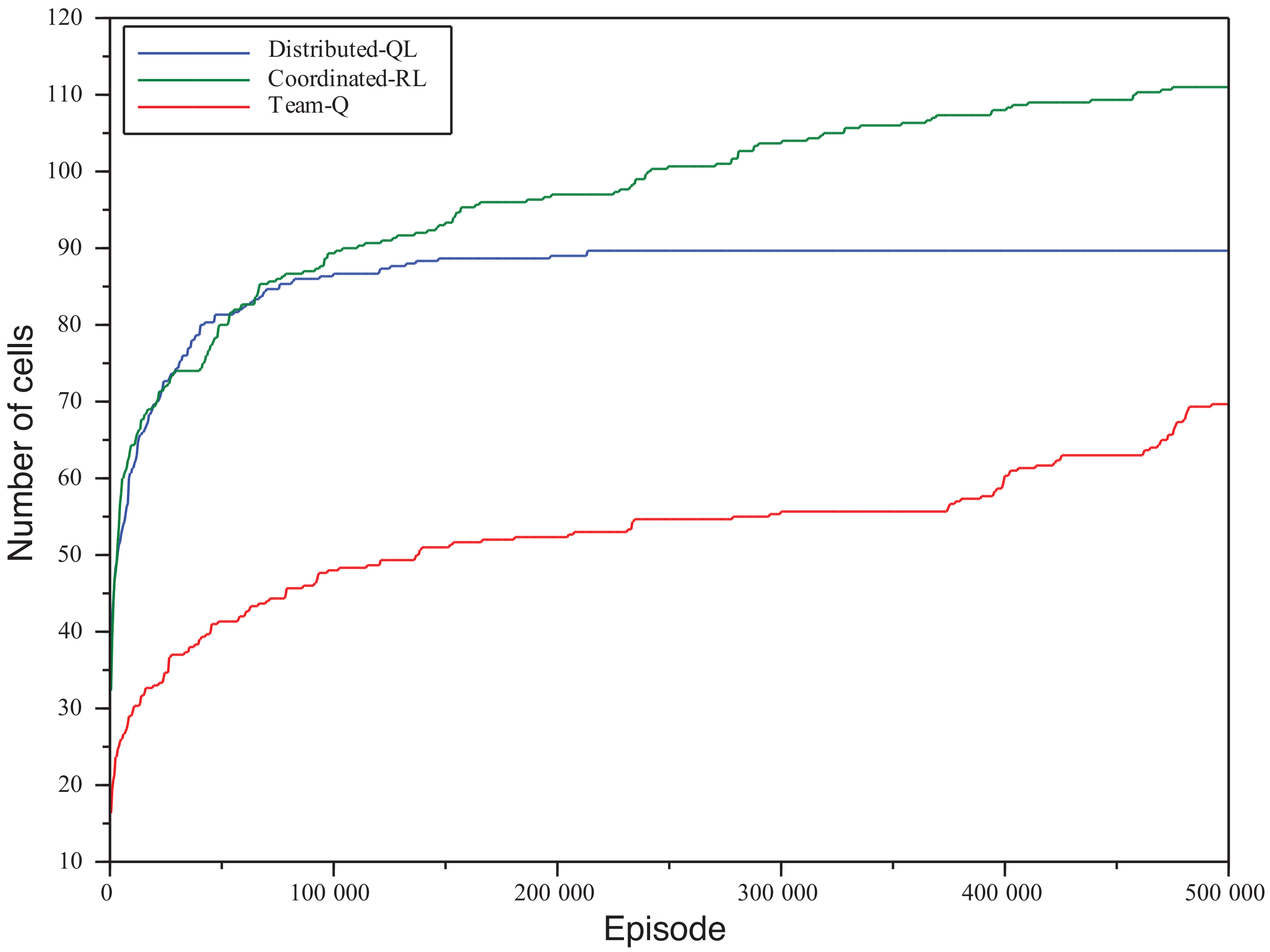

Fig 8. Number of cells reached with the tip of the hose with $n_e = 5 \cdot 10^5$, separate plots when applying Distributed-QL, Coordinated-RL, and Team-QL algorithms.

doi:10.1371/journal.pone.0127129.g008

stochastic game (C-SG) from the local policies learned by D-RR-QL. We give a consensus-based implementation for the decentralized determination of RR turns that may allow fully distributed implementation of the D-RR-QL.

Computational experiments show that D-RR-QL using MSAV policies can provide a valid joint-action policy approximating the optimal policy faster than D-QL, Team-QL, and Coordinated-RL in over-constrained systems. Furthermore, the D-RR-QL convergence is not limited to deterministic MDPs, it can also cope with stochastic environments. Communication requirements to learn local Q-values ($O(1)$ messages each step if communications are used, 0 messages otherwise) and to obtain the global policy deciding a joint-action (each action requires $O(N)$ messages) are minimal when compared to coordinated multi-agent approaches such as Coordinated-QL. As a line of future work, we need to study the equivalence between C-SG and C-RR-SG to determine the degree of approximation that can be obtained when there

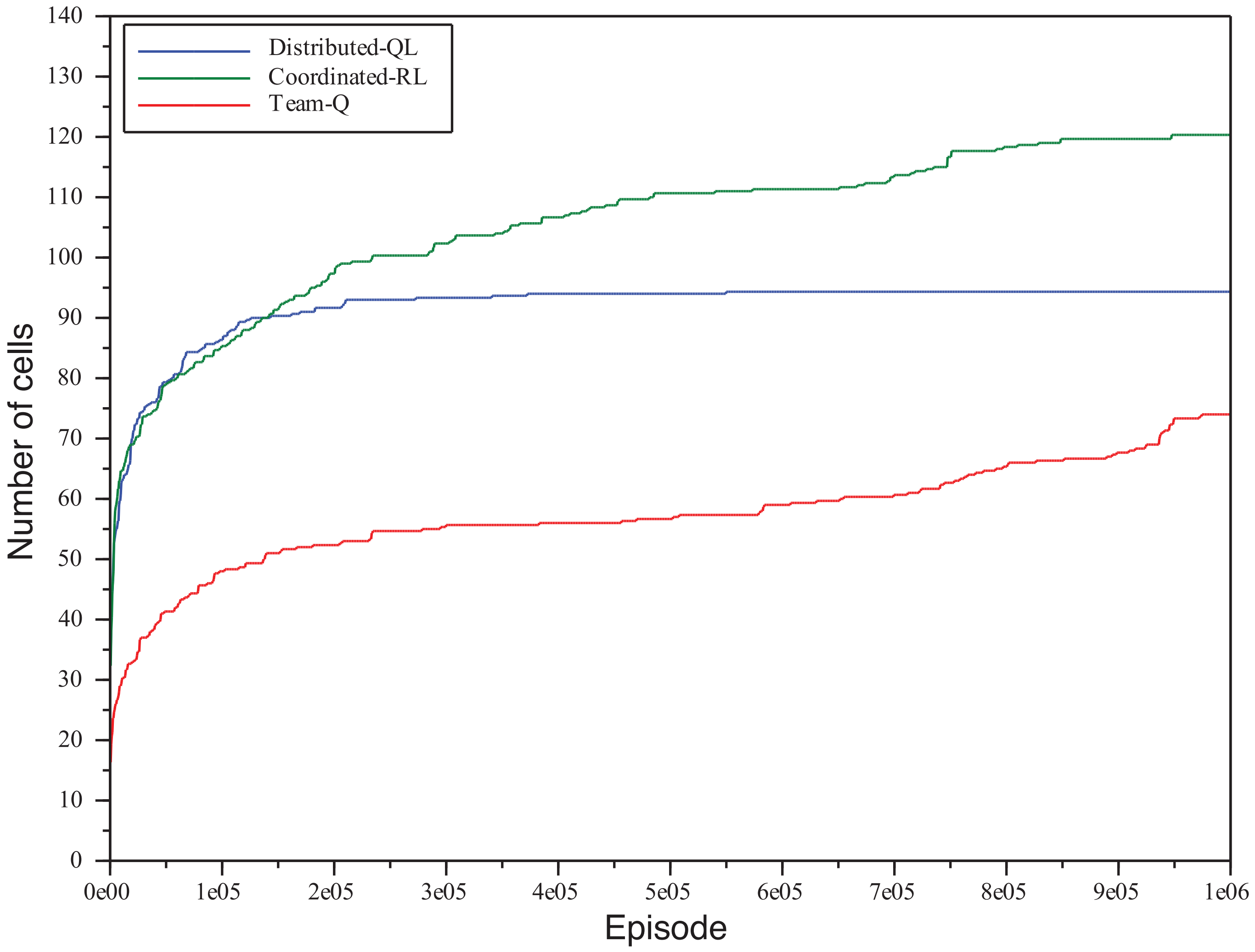

Fig 9. Number of cells reached with the tip of the hose with $n_e = 10^6$, separate plots when applying Distributed-QL, Coordinated-RL, and Team-QL algorithms.

doi:10.1371/journal.pone.0127129.g009

is no C-RR-SG equivalent to a C-SG. We also plan to investigate more general ways to learn how to avoid reaching undesirable terminal states. On the other hand, the degree of abstraction used in our experiments limits the applicability to real robot control problems. This approach can be directly applied in real environment as long as low-level controllers and sensors provide this kind of abstract actions and states, but the coarseness of the state-action space representation limits the optimality of the learned policies. Therefore, we plan to use continuous state-action spaces in our future research.

Author Contributions

Conceived and designed the experiments: BF IE MG. Performed the experiments: BF. Analyzed the data: BF IE MG. Wrote the paper: BF MG.

References

1. Duro R, Graña M, de Lope J (2010) On the potential contributions of hybrid intelligent approaches to multicomponen robotic system development. Information Sciences 180: 2635–2648. doi: 10.1016/j.ins.2010.02.005
2. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: A survey. Journal of Artificial Intelligence Research 4: 237–285.
3. Sutton R, G Barto AG (1998) Reinforcement Learning I: Introduction. MIT Press. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi = 10.1.1.32.7692.
4. Watkins C (1989) Learning from delayed rewards. Ph.D. thesis, University of Cambridge, Psychology department. URL http://www.cs.rhul.ac.uk/ ~ chrisw/thesis.html.
5. Watkins C, Dayan P (1992) Technical note: Q-learning. In: Machine Learning. volume 8, pp. pp. 279–292. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.8118&rep = rep1&type = pdf.
6. Fernandez-Gauna B, Lopez-Guede JM, Zulueta E, Echegoyen Z, Graña M (2011) Basic results and experiments on robotic multi-agent system for hose deployment and transportation. International Journal of Artificial Intelligence 6: 183–202.
7. Fernandez-Gauna B, Lopez-Guede JM, Zulueta E, Graña M (2010) Learning hose transport control with Q-learning. Neural Network World 20: 913–923.
8. Fernandez-Gauna B, Lopez-Guede JM, Etxeberria-Agiriano I, Ansoategi I, Graña M (2014) Reinforcement learning endowed with safe veto policies to learn the control of l-mcrs. Information Sciences in press.
9. Busoniu L, Babuska R, De Schutter B (2008) Comprehensive survey of multiagent reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews 38: pp. 156–172. doi: 10.1109/TSMCC.2007.913919
10. Arel I, Liu C, Urbanik T, Kohls A (2010) Reinforcement learning-based multi-agent system for network traËšc signal control. Intelligent Transport Systems, IET 4: 128–135. doi: 10.1049/iet-its.2009.0070
11. Zhao G, Sun R (2010) Application of multi-agent reinforcement learning to supply chain ordering management. In: Natural Computation (ICNC), 2010 Sixth International Conference on. volume 7, pp. 3830–3834. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber = 5582551.
12. Tamakoshi H, Ishii S (2001) Multiagent reinforcement learning applied to a chase problem in a continuous world. Artificial Life and Robotics 5: 202–206. doi: 10.1007/BF02481502
13. Servin A, Kudenko D (2008) Multi-agent reinforcement learning for intrusion detection. In: Tuyls K, Nowe A, Guessoum Z, Kudenko D, editors, Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning, Springer Berlin / Heidelberg, volume 4865 of *Lecture Notes in Computer Science*. pp. 211–223. URL http://dx.doi.org/10.1007/978-3-540-77949-0_15.
14. Melo F, Ribeiro M (2010) Coordinated learning in multiagent mdps with infinite state-space. Autonomous Agents and Multi-Agent Systems 21: 321–367. doi: 10.1007/s10458-009-9104-y
15. Tadepalli P, Ok D (1996) Scaling up average reward reinforcement learning by approximating the domain models and the value function. In: In Saitta. Morgan Kaufmann, pp. 471–479. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.1556&rank=1.
16. Bowling M, Veloso M (2002) Scalable learning in stochastic games. In: In: AAAI Workshop on Game Theoretic and Decision Theoretic Agents. pp. 11–18. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.6415.
17. Sutton RS (1996) Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Advances in Neural Information Processing Systems 8. MIT Press, pp. 1038–1044. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.4764.
18. Crites R, Barto A (1996) Improving elevator performance using reinforcement learning. In: Advances in Neural Information Processing Systems 8. MIT Press, pp. 1017–1023. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.5519&rank=1.
19. Xiao H, Liao L, Zhou F (2007) Mobile robot path planning based on q-ann. In: Automation and Logistics, 2007 IEEE International Conference on. pp. 2650–2654. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4339028&tag=1.
20. Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. In: Machine Learning. pp. 37–66. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.138.635&rank=10.
21. Kok JR, Vlassis N (2006) Collaborative multiagent reinforcement learning by payoË™ propagation. Journal of Machine Learning Research 7: 1789–1828.
22. Boyan JA, Moore AW (1995) Generalization in reinforcement learning: Safely approximating the value function. In: Advances in Neural Information Processing Systems 7. MIT Press, pp. 369–376. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.7599.

23. Lauer M, Riedmiller MA (2000) An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In: Proceedings of the Seventeenth International Conference on Machine Learning. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., ICML'00, pp. 535–542. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.772.

24. Littman ML (2001) Value-function reinforcement learning in markov games. Cognitive Systems Research 2: 55–66. doi: 10.1016/S1389-0417(01)00015-8

25. Guestrin C, Lagoudakis M, Parr R (2002) Coordinated reinforcement learning. In: In Proceedings of the IXth ICML. pp. 227–234. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.5626.

26. Theetten A, Grisoni L, Andriot C, Barsky B (2008) Geometrically exact dynamic splines. Computer-Aided Design 40: 35–48. doi: 10.1016/j.cad.2007.05.008

27. Echegoyen Z, Villaverde I, Moreno R, Graña M, d'Anjou A (2010) Linked multi-component mobile robots: Modeling, simulation and control. Robotics and Autonomous Systems 58: 1292–1305. doi: 10.1016/j.robot.2010.08.008

28. Fernandez-Gauna B, Marques I, Graña M (2013) Undesired state-action prediction in multi-agent reinforcement learning. application to multicomponent robotic system control. Information Sciences 232: 309–324.

29. Fernandez-Gauna B, Lopez-Guede J, Graña M (2013) Transfer learning with partially constrained models: application to reinforcement learning of linked multicomponent robot system control. Robotics and Autonomous Systems 61: 694–703. doi: 10.1016/j.robot.2012.07.020

30. Claus C, Boutilier C (1997) The dynamics of reinforcement learning in cooperative multiagent systems. In: In Proceedings of the Fifteenth National Conference on Artificial Intelligence. AAAI Press, pp. 746–752. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.

31. Schneider J, Wong WK, Moore A, Riedmiller M (1999) Distributed value functions. In: In Proceedings of the Sixteenth International Conference on Machine Learning. Morgan Kaufmann, pp. 371–378. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.898.

32. Czibula G, Bocicor MI, Czibula IG (2011) Solving the protein folding problem using a distributed q-learning approach. International Journal of Computers 5: 404–413.

33. Bocicor MI, Czibula G, Czibula IG (2011) A distributed q-learning approach to fragment assembly. Studies in Informatics and Control 20: 221–232.

34. Matignon L, Laurent G, Le Fort-Piat N (2007) Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In: Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on. pp. 64–69. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4399095.

35. Mariano CE, Morales E (2000) A new distributed reinforcement learning algorithm for multiple objective optimization problems. In: In Memorias del Segundo Encuentro Nacional de Computaci on ENC99, paper No. 111. pp. 12–15. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.5216.

36. Kapetanakis S, Kudenko D (2002) Reinforcement learning of coordination in cooperative multiagent systems. In: AAAI/IAAI 2002. pp. 326–331. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.3184.

37. Wang X, Sandholm T (2002) Reinforcement learning to play an optimal nash equilibrium in team markov games. In: in Advances in Neural Information Processing Systems. MIT Press, pp. 1571–1578. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.2669&rep = rep1&type = pdf.

38. Guestrin C, Koller D, Parr R (2001) Multiagent planning with factored mdps. In: NIPS-14. The MIT Press, pp. pp. 1523–1530. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.9428.

39. Dietterich T (2000) An overview of maxq hierarchical reinforcement learning. In: Choueiry B, Walsh T, editors, Abstraction, Reformulation, and Approximation, Springer Berlin / Heidelberg, volume 1864 of *Lecture Notes in Computer Science*. pp. pp. 26–44. URL http://dx.doi.org/10.1007/3-540-44914-0_2.

40. Hengst B (2004) Model approximation for hexq hierarchical reinforcement learning. In: In ECML 2004. pp. pp. 144–155. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.6505.

41. Cheng X, Shen J, Liu H, Gu G (2007) Multi-robot cooperation based on hierarchical reinforcement learning. Lecture Notes in Computer Science 4489: 90–97.

42. Menache I, Mannor S, Shimkin N (2002) Q-cut: Dynamic discovery of sub-goals in reinforcement learning. In: Elomaa T, Mannila H, Toivonen H, editors, Machine Learning: ECML 2002, Springer Berlin / Heidelberg, volume 2430 of *Lecture Notes in Computer Science*. pp. pp. 187–195. URL http://www.springerlink.com/content/73j0ndvpwrt2mkhp/.

43. Geibel P, Wysotzki F (2005) Risk-sensitive reinforcement learning applied to control under constraints. J Artif Int Res 24: 81–108.

44. Geibel P (2006) Reinforcement learning for mdps with constraints. In: ECML. pp. 646–653. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.83.1601.

45. Geibel P (2001) Reinforcement learning with bounded risk. In: In Proceedings of the Eighteenth International Conference on Machine Learning. Morgan Kaufmann, pp. 162–169. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.8802.

46. Abbeel P, Ng AY (2005) Exploration and apprenticeship learning in reinforcement learning. In: in Proc. 21st International Conference on Machine Learning. ICML, pp. 1–8. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.74.2108.

47. Hans A, Schneegaß D, Schäfer AM, Udluft S (2008) Safe exploration for reinforcement learning. In: ESANN. pp. 143–148. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.161.2786.