Cooperative Multi-Agent Reinforcement Learning for Multi-Component Robotic Systems: guidelines for future research

Manuel Graña, Borja Fernandez-Gauna, Jose Manuel Lopez-Guede

July 2, 2011

Abstract

Reinforcement Learning (RL) paradigm aims to develop algorithms that allow to train an agent to optimally achieve a goal with minimal feedback information about the desired behavior, which is not precisely specified. Scalar rewards are returned to the agent as response to its actions endorsing or opposing them. RL algorithms have been succesfully applied to robot control design. The extension of the RL paradigm to cope with the design of control systems for Multi-Component Robotic Systems (MCRS) poses new challenges, maily related to coping with scaling complexity up due to the exponential state space growth, coordination issues, and the propagation of rewards among agents. In this paper, we identify the main issues which offer oportunities to develop innovative solutions towards fully-scalable cooperative multi-agent systems.

1 Introduction

Multi-Component Robotic Systems (MCRS) [22] are currently the focus of great scientific interest because they are expected to provide solutions to the growing set of applications that require groups of autonomous robots able to dynamically adapt to changing environments and act in a coordinated way to perform tasks that could not be performed by a single robot, or performing them in a more efficient or economical way. Examples of such tasks are cooperative mapping of an environment [2], establishing dynamic communication links, and driving a hose to a goal [23, 24]. Among the desired properties of a MCRS control algorithm, the most salient are:

- Resource scalability. Applications may require teams of robots of different sizes depending on task specific parameters. The addition of new robots must not degrade the operation of the whole system, implying that the resources (memory and communication bandwidth) used by the control algorithm must not grow exponentially.

- Automating system design. It is not desirable to rely the success of a system on the expertise of the designer and it is preferable to develop tools that can automatically tailor the system to new complex environments. Automatic decomposition of complex tasks allows the definition of the robot team coordination as workload distribution. .

- Heterogeneity. MCRS applications may include heterogeneous groups of robots with different capabilities, including both sensors and actuators. Thus, control algorithms should be able to deal with heterogeneous inputs and outputs, and also minimize the impact of less performing robot individuals on the whole system's performance.

- Decentralized control to obtain higher fault-tolerance than with centralized control. Decentralized control improves scalability because control complexity grows linearly with the number of robots.

- Accurate control. Some mechanism is required to compensate for the inherent delay introduced by the sensory-devices, control algorithm and communications.

- Robustness to partial and noisy sensor data. In complex and noisy environments it is a requirement that agents are able to carry on their tasks even if they only have inaccurate and partial knowledge of the environment.

- Reasonable development time. Designing and fine-tuning the control algorithm should require an affordable amount of time.

Often, developing control algorithms for MCRS cannot be approached analytically. Sometimes there is not even a proper model for the system to be controlled and, even when such a model is available, system complexity hind their analytical resolution. As an alternative to traditional control theory, Artificial Intelligence techniques have been explored to provide robotic systems with tools that enable them to learn by interacting with the environment, which can be either the real world or a simulated one.

Reinforcement Learning (RL)[66] has been succesfully applied to develop control policies for robotic systems in the recent past. RL algorithms deal with learning how to maximize accumulated rewards received in a trial-and-error fashion. This learning problem has mainly been approached using three different families of algorithms: Dynamic Programming, Monte-Carlo and Time-Difference methods. Agents are the subject of learning processes. Equating agents to robots, training of decentralized control of MCRS can be viewed as an instance of Multi-Agent Reinforcement Learning (MARL) systems. We will consider, cooperative Multi-Agent systems designed to maximize the collective utility of the system as a whole. On the other hand, competitive systems are designed such that each agent only intends to maximize its individual utility[32, 50]. In fact, there is almost no literature on the application of MARL to learn

the control of MCRS-like systems, therefore the orientation of this paper is towards the identification of promising lines of research.

Scalability refers to the ability of MARL to cope with growing size of the system and environment. In MCRS this size is directly related to the number of robots in the system. The need to provide a unified representation for the states of the individual robots and their action policies implies a combinatorial explosion of the state-action space which forbids some MARL approaches to MCRS control learning.

This paper is structured as follows: we first give some background on RL in Section 2. We review some of the main issues applying MARL methods to the control of MCRS in Section 3. Section 4 identifies the main approaches towards the development of scalable MARL systems that can lead to innovative solutions of the MCRS control problem. Finally, Section 5 gives our thoughts about some of the work yet to be done before they become a general solution to real complex environments.

2 Reinforcement Learning

Single-Agent RL methods model systems as Markov Decision Processes (MDPs), defined by a tuple (S, A, P, R), where S is a finite set of states, A is a set of actions from which the agent can choose, $P : S \times A \times S \rightarrow [0, 1]$ is a transition function $P(s, a, s')$ that defines the probability of observing state s' after executing action a in state s, and $R : S \rightarrow \mathbb{R}$ is the expected reward after taking action a in state s. The policy applied by the agent to select the action performed at each state is modeled in stochastic environments as a probability distribution $\pi : S \times A \rightarrow [0, 1]$ giving the probability of taking action a in state s. The goal of the agent is to maximize the accumulated rewards received.

Almost all RL algorithms estimate the *value* of being in a state s (it can alternatively be viewed as the value of taking action a in state s), as the expected accumulated rewards received by the agent from that state following policy π. This estimation of the state value, denoted $V^{\pi}(s)$, can be expressed as:

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

where s_t and r_t are the observed state and reward at time-step t, $E_{\pi}\{.\}$ denotes the expected value from time step t given that the agent follows policy π thereafter, and $\gamma \in [0, 1]$ is a discount-rate parameter that penalizes lengthy sequences of actions by weighting early rewards higher than later ones, dampening the value returned from late actions. There exists always one or more optimal policies π^* that maximize the expected state value, and all share a common optimal value function V^* satisfying:

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

Similarly, the value of taking an action a in state s is usually estimated using the state-action value function $Q^{\pi}(s,a)$, which can be written as

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

where a_t represents the action taken at time-step t. The optimal state-action pair is, therefore,

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

An action selection policy π must also be defined to realize an MDP. While the straightforward approach (*greedy* action selection) involves always selecting the action with the highest Q-value, thus *exploiting* all available knowledge, this prevents the agent from *exploring* yet unknown *action-state* pairs and thus, hinders it to discover potentially better actions. The compromise between *exploration* and *exploitation* is solved using either a $\epsilon - greedy$ algorithm (a random action is selected with probability ϵ while the best action is chosen with probability $1-\epsilon$) or a Soft Max action selection based on a Boltzmann distribution:

$$\pi(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}}, \tag{1}$$

where τ is a positive *temperature* parameter, low values of τ increase the probability of taking actions with high Q-values, high-temperatures yield random action selections. Reinforcement Learning (RL) deals with the discovery of the optimal policy from the interaction between the MDP and its environment by means of the rewards that the MDP receives because of its actions.

Dynamic Programming-based RL algorithms require complete knowledge of probability distributions of all possible state transitions, therefore this requirement limits their applicability to complex real environments. On the other hand, Monte-Carlo and Time-Difference methods need only a model that provides sample state transitions making these algorithms able to learn on-line. Moreover, while Monte-Carlo methods learn after a finite amount of experience is finished, Time-Difference methods can learn on a single time-step basis.

One of the best understood and most widely used Time-Difference algorithms is Q-Learning, discovered by Watkins [74]. The original tabular Q-Learning algorithm estimates the value of a state s' on a one-step look-ahead fashion: $\max_{a'}\{Q(s',a')\}$, endowing the agent with the ability to learn on a one-step basis. This table is updated according to the following expression:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma * \max_{a'} Q(s',a') - Q(s,a)\right), \tag{2}$$

where $\alpha \in [0,1]$ is an step-size parameter indicating how fast the agent is desired to learn. In [74] convergence of Q-learning in a stationary environment to an

optimal policy with probability 1 is proved, as long as all state-actions pairs keep being updated. This is a theoretically sensible condition, but hard to meet in practice because an agent may not be able to explore sufficient space to guarantee convergence. To relax this condition, Greedy in the Limit with Infinite Exploration (GLIE) policies were proposed [61].

Initial approaches considered learning in a MCRS with n units as a unique learning process (a single agent) that had access to all environment variables and could control all robots applying simultaneous joint actions $A^n \equiv \{a_1, \ldots, a_n\}$, where $a_i \in A$ denotes the action applied by the i^{th} robot. This kind of learning systems are known as *team learning* [50] and are not scalable to big robot teams for obvious reasons: the size needed to store the Q-table grows exponentially with the number of agents, even if we consider that the state space does not grow, because the action-state space size order is $O(\mid S \times A^n \mid)$. Besides, centralized control is less fault-tolerant than distributed control. *Concurrent learning* considers the presence of multiple *agents* implying that each of them is entitled to select its own actions and learn for itself how to maximize its local reward function. We have then an instance of Multi-Agent RL (MARL) [11]. In the cooperative MARL, a shared *global reward* is used as a quality assessment of the whole system behavior. A naive approach to reduce the storage requirements is limiting the environment information available to each agent, expecting that maximizing local rewards will maximize the global reward, but additional coordination mechanisms are usually required.

3 Issues of Multi-Agent Reinforcement Learning for MCRS control

The main advantage of adopting the MARL framework for the development of MCRS control algorithms is that the definition of the MDP modeling the system whose control is to be learnt is a systematic way to deal with the problem, compared with ad-hoc designing and developing a control algorithm (even using supervised learning methods). However, applying MARL algorithms to MCRS raises several strong issues. Coordination-related issues are specific to MARL algorithms, others are inherited from the basic single-agent RL methods, which only may get worse in multi-agent configurations because of the added complexity of the system.

- *Resource scalability:* The intractable growth of memory requirements is the most serious limitation of the tabular representation of Q-matrices. In single-agent problems the size order of the table is $O(\mid S \times A \mid)$ and this gets even worse in most MARL algorithms, growing exponentially as the number of agents increase: $\mid S \times A^n \mid$. This problem is known as the *curse of dimensionality* and is the most serious limit to scale up the single agent RL Q-Learning. Besides, communication resources needed for RL also scale up combinatorially with the number of robots/agents. Hierarchical solutions [26] can be considered to face this problem. Single-agent RL

requires chosen actions to be transmitted to all agents at each time-step and multi-agent explicit-coordination mechanisms require even a bigger communication bandwidth for the agents to agree on a joint action.

- *Action Heterogeneity:* In standard RL formulation all actions span the same fixed amount of time. This is not a very realistic assumption in MCRS. Action are abstractions of operations performed with different electronic devices which usually require different amounts of time to perform equivalent actions. For example, if an action consists in the motion across a length of space, heterogeneous robotic units could require different amounts of time to complete the action. Furthermore, different actions may require wide differences in time in the same robotic unit, i.e. moving versus switching on/off a LED. Dealing with this time dimension means adding the complexity of synchronization on top of coordination.

- *Decentralized control:* A major issue towards achieving multi-agent coordination through MARL is that the environment becomes non-stationary from the individual agent point of view because other agents' policies will change during the learning process. This is likely to produce oscillations and unexpected behaviors. This problem has been extensively studied as an *Stochastic Game*, leading to the concept of *Nash Equilibrium* [11]. If each agent follows an optimal policy relative to other agents' optimal policies, then the system is said to have reached Nash Equilibrium. However, there may exist more than one optimal policy achieving Nash equilibrium.

- *Control delay:* All on-line RL algorithms follow the same iterative pattern: observe the state, select an action and then issue the appropriate command to the actuators. This is completely safe in an ideal scenario where acquiring the state, executing the action selected and transmitting the command introduces no time delay, but in real life observation, communication and decision consume time and add complexity to the synchronization issue. I.e. coordination algorithms [29, 39] introduce complex communication protocols to agree on a joint action to be taken.

- *Robustness to partial and noisy sensor data:* Most approaches assume omniscient agents aware of all the sensed information, but this approach is unrealistic in complex environments facing serious limitations , i.e. physically limited and error-prone communications, sensor physical limitations and/or obstacles. Furthermore, noisy measurements are likely to be perceived as different states in multi-agent environments. Therefore, algorithms that maximize the success possibilities in the presence of incomplete and noisy data are desired.

- *Convergence time:* Before the MCRS can be effectively controlled, the RL algorithm must explore the state-action space. The time required for this learning process can be unaffordable in real applications with large state-action spaces and thus, methods for a faster on-line learning are desirable.

MARL systems may require even greater learning time because of the coordination requirements introduced.

4 Avenues for research

In this section we identify approaches found in the literature that may be sources of innovative solutions for the MCRS control problem, overcoming the complexity explosion of such systems. The main categories of these approaches deal with alternative system models, valuation functions, task decomposition and ways to structure the learning process.

4.1 Alternative System Models

Several modeling enrichments developed in the context of single-agent RL can be extended to MARL in order to cope with some specific features of MCRS or with its inherent scaling problem. These enrichments are focused in the state identification or in the modeling of actions. They propose variations of the basic MDP underlying the RL.

4.1.1 Modeling action duration

The basic MDP model assume actions to have the same duration, preventing the use of abstract or heterogeneous actions such as *Open-the-door* or *Move-East*. Semi-Markov Decision Processes (SMDPs) were introduced [52, 43, 10] to allow the definition of actions that may take different amounts of time to finish. Denoting $N(a)$ the number of time-steps required by an action a , the duration dependent transition and reward functions can be reformulated as $P(s, a, s', N(a))$ and $R(s, N(a))$, respectively. This SMDP model only considers indivisible variable-length actions and does not provide any way to model the nature of these timed actions (also called *macro-actions*, *abstract actions* or *sub-controllers* in the literature).

Sutton et al. propose in [64] a more general framework defining *Markov Options* as a generalization of primitive actions that have three components: a policy $\pi : S \times A \to [0, 1]$, a termination condition $\beta : S^+ \to [0, 1]$, and an initiation set $I \subseteq S$, where S^+ represents the regular states plus the terminal states. To handle optional timeouts, *Semi-Markov Options* allow to model termination conditions and policies which may not only depend on the current state s_{t+k} but on the whole sequence of states observed since the Markov option started in state s_t: $(s_t, a_t), (s_{t+1}, a_{t+1}) \ldots, s_{t+k}$. *Semi-Markov Options* are therefore defined by a policy $\pi : \Omega \times A \to [0, 1]$, a termination condition $\beta : \Omega \to [0, 1]$, where Ω denotes the set of state sequences. The set of selectable options at any given state s is denoted $\mathcal{O}_s$ and the whole set of options is thus $\mathcal{O} = \bigcup_{s \in S} \mathcal{O}_s$. The approach allows defining policies over options: $\mu : S \times \mathcal{O} \to [0, 1]$.

This modeling framework is very appealing, offering a huge set of possibilities, such as to abort an option if a better one is available, and to define

sub-goals considering transitions between sub-goals as sub-problems easier to learn. This could be of direct application to the modeling of synchronization situations in MCRS control, when some robot units must wait until some condition is accomplished by other units. However, the programmer is expected to provide a complete set of policies, which can be a hard task. The approach is not easy to scalable to large and complex problems.

4.1.2 Partially Observable Models

The focus of partially observable models is the unability to have complete knowledge of the system state, so that the process must be guided by the partial knowledge provided by an observation function which returns measurements that can be used to learn policies despite ignorance of the full state effect of the actions.

A Partially Observable MDP (POMDP) [36] is defined as tuple (S, A, P, R, Ω, O), where the tuple (S, A, P, R) describes a MDP, Ω is a finite set of past environment observations the agent has made, and $O : S \times A \to \Pi(\Omega)$ is the observation function, specifying a probability distribution over possible observations such that $O(s', a, o)$ is the probability of making observation o given that the agent took action a reaching state s'. No distinction is made in this model between actions meant to change the environment or to observe it, and belief estimations are used to take decisions. Decentralized-MDP (DEC-MDP) and Decentralized-POMDP (DEC-POMDP) models respectively extend MDP and POMDPs to the cooperative multi-agent case using a global reward, but this kind of systems is known to be very hard to scale because of their NEXP-complete complexity[7], and only a Dynamic Programming algorithm has been proved to optimally solve them[6]. DEC-POMDP with Communication further yet expands this model immediate and costly communications, communication decisions and rewards depending on communications in the model. Estimating the whole environment state from a set of observed measurements has also been approached as a generalization problem, i.e. using Recurrent Neural Networks[58].

4.1.3 Automatic State Abstraction

Automated state abstraction approaches consider the problem of aggregating states into state partitions that share some common properties. This approach tries to cope with the combinatorial explosion of the state space through a hierarchical decomposition approach. Early work in automatic state abstraction include statistical t-test analysis to measure the relevance of binary state variables [12] and soft-aggregation methods to map state projections [62, 47]. Fuzzy theory has also been applied to obtain abstractions of state sets and generalize over them[4, 5, 21]. Some authors have also empirically studied different manually set state abstraction operations, such as [25] which studied symmetry and multi-agency homomorphic mappings. Homomorphisms may allow to reduce the size of MDPs, but they do not guarantee that the reduced problem is relevant to solve the original one. [42] proposed a unified theoretical framework

to define abstractions and studied some properties of five different abstraction operations, giving some interesting insights into their respective benefits and limitations. This approach can be of use for MCRS because the state space naturally partitions into the local states of the robots, plus some variables modeling coordinations/synchronization processes.

In robotic applications, this procedure leads to the partition of the configuration-action space into continuous compact regions of similar or equivalent rewards in the sense of contributing to the fulfillment of the assigned task. A notion of equivalence based on *bi-simulation* is introduced in [27]. The authors propose to aggregate states that are both action sequence equivalent and optimal value equivalent. An algorithm is proposed to optimally reduce a MDP to an equivalent one so that the optimal policy over the reduced MDP is also the optimal policy for the original model.

Another interesting approach consists in defining some state variable relevance criterion [33]. Assuming that the state space S is the cartesian product $S = X \times Y$ of the state variable sets $X = \{X_1, ..., X_n\}$ and $Y = \{Y_1, ..., Y_m\}$, $[s]_X$ is defined as the projection of S onto X and using $s' \models [s]_X$ to denote that s' agrees with s on every state variable in X, Y is said to be *policy irrelevant* if an optimal policy is optimal for both the original state space and the projected one, formally: $\exists a; \forall s' \models [s]_X ; \forall a'; Q^*(s', a) \geq Q^*(s', a')$. A statistical hypothesis test is proposed to determine how relevant a state variable is, but it requires an optimal value function. [14] proposes to measure the variance of the value function among states that only differ in the value of one state variable, therefore estimating the relevance of variable states before the actual value function is available. This is particularly interesting for task decomposition approaches, because using only subtask-relevant variables can further reduce the complexity of subtasks, yielding higher scalability.

4.2 Value Function Approximation

The most straight-forward approach to avoid the exponential growth of the storage requirements in Q-Learning is to use a *Value Function Approximator* (VFA) instead of the tabular representation of the value function Q. These approaches provide generalizations of available experience estimating a response to yet unobserved states, and some of them involve also abstraction, for they need not store the observed experience after the VFA is updated (trained) accordingly to it. Many general-purpose function approximators have been reported to build VAFs: Local Linear Regression [67], weighted Radial Basis Functions [40], Cerebellar Model Arithmetic Computers [8, 65], Artificial Neural Networks [17, 76], instance-based approximators[1], and Least Squares Policy Iteration[29]. It has been discussed whether VAFs might be appropriate in the general case (in favor [65], against [9]), because of the assumptions on the topology of the functions [63]. They remain to be applied to MARL systems. VAF approaches can be directly related to state aggregation, because they can be defined on the aggregated values providing a hierarchical evaluation of the value function.

4.3 Automatic Task Decomposition

After manually designed task decomposition was successfully applied to increasingly complex environments[53], the automatic decomposition of tasks became a hot subject and it has thereafter focused great scientific interest [15], because of the inherent scalability of automated approaches.

A medium level of automation is introduced in [59], which proposes defining some basic MAXQ hierarchy to introduce domain knowledge in the system and using options to learn subtasks in some hierarchy level. After constructing a transition-graph, vertices are clustered using an artificial immune network model until a preset number of clusters (options) are discovered.

Another approach is *HEX-Q* [31], a method that automatically discovers hierarchies in single-agent RL problems by finding repeated sub-structures, but is limited to work in environments meeting three conditions: (a) some of the variables in the state vector change less frequently than others, (b) variables that change more frequently retain their transition properties in the context of the more persistent variables and (c) the interface between regions can be controlled. This is most likely to work in structured environments, such as buildings with different number of floors and rooms, and requires a coherent representation of state variables. The algorithm first constructs a Directed Graph and clusters the states by the less often changing state variables (i.e. the floor), then decomposes it in Strongly Connected Components (SCC), which are further combined to recursively form regions maximizing their size. States that are part of trajectories between different regions are labeled as *exits* and *entries*.

More general approaches [13, 45, 48, 60] are based on transition-graphs partitioning techniques. First, the state space is randomly explored while storing the history of observed state-transitions, then a transition-graph is built from transition history, usually building a directed graph $G = (V, E)$, where set V is the set of vertices representing states ($V \in S$) and E is the set of edges (s, s') representing observed transitions $s \rightarrow s'$ between states. Defining subtasks in a transition-graph is commonly considered as a clustering problem, implicitly assuming state clusters are considered subtasks to be solved towards reaching sub-goals, which have been identified as states with a high reward gradient [20] or states that are often visited on successful trajectories [46], but mostly, sub-goals are identified as bottlenecks (such as a door separating two rooms) between densely connected state clusters. Reaching a sub-goal state or region of states is usually considered a subtask. The *Q-Cut* algorithm [48] was proposed using the Min-Cut procedure to partition the directed graph using network flow analysis. This partition algorithm has complexity $O\left(m^3\right)$, where m is the number of nodes or states, and uses the complete transition-graph meaning it doesn't scale well to the number of states. *Relative Novelty* (RN)[3] was proposed as a means of overcoming this limitation. It only considered the last observed transitions, thus bounding m, and even more important, its execution complexity on the number of nodes is $O\left(1\right)$. The downside is the use of parameters that need to be set heuristically. *L-Cut* [60] was presented as a more scalable par-

titioning algorithm than *Q-Cut* and, just as RN, it only considered part of the state transitions. To measure the quality of a binary partition, a normalized cut metric was chosen (*NCut*) and, because computing the partition that minimizes NCut is a NP-hard computational problem, the algorithm approximated this metric for every $m-1$ possible binary partition by using spectral graph theory. The complexity is $O\left(th^3\right)$ where t is the number of transition samples and h the number of nodes of the local transition-graph (note that $h \ll m$). Towards a fully-automated process, [14] proposed the use of the smoothness property of the second smallest eigenvector of the Laplacian to recursively partition the transition-graph until a predefined number of clusters is reached. [45] proposed using small sets of states rather than unique states as sub-goals and presented two heuristic methods for clustering: (a) by topology: given a preferable size of clusters, their quality is proportional to the size of the smallest cluster and (b) by value: states are clustered so the value differences are minimized in each cluster.

Recently, Dynamic Bayesian Networks (DBN) have been proposed to approximate state transition probabilities of a factored MDP. They presented the *Variable Influence Structure Analysis* (VISA)[34, 35] algorithm, that following some of the concepts of the HEX-Q approach, decomposed factored MDPs into SCC and was able to neglect state variables irrelevant to an SCC. This algorithm requires the existence of two or more SCC and that a DBN is given, which cannot be assumed in most problems.

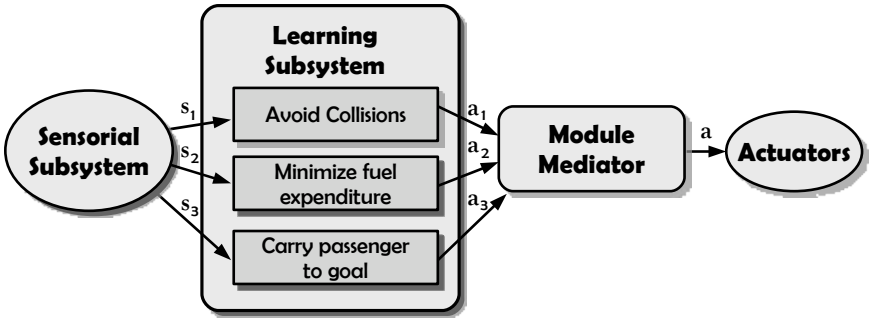
Task decomposition has also been approached using Diverse Density to solve a Multiple-Instance Learning Problem[46], identifying sub-goals as small sets of states often visited in successful trajectories.

4.4 Structured Single Agent Reinforcement Learning

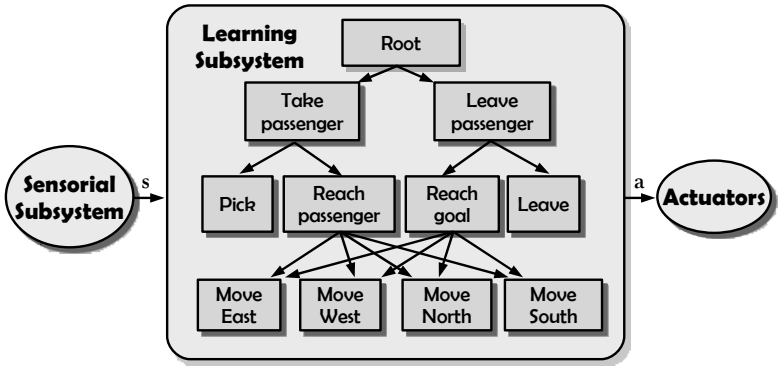
Learning processes can be simplified by decomposing tasks into more manageable subtasks, thus reducing the original problem complexity. This decomposition has the additional advantage of reducing the total amount of information required to solve the problem, if subtasks are appropriately defined. There are two main Structured RL approaches: Modular Reinforcement Learning (MRL) and Hierarchical Reinforcement Learning (HRL). The former considers executing concurrent subtasks, while the latter defines a hierarchical structure of tasks. To illustrate the approaches we will consider this variant of the classical *taxi driver problem* [18]: a taxi driver has to transport a passenger from his current location to some predefined goal position and can choose an action $a \in A$, where A=$\{$*Pick*, *Leave*, *Move-North*, *Move-East*, *Move-South*, *Move-West*$\}$.

4.4.1 Modular Reinforcement Learning

A modular approach involves learning different subtasks or behaviors concurrently and using a *Module Mediator* (also referred to as *Module Arbiter* in the literature) responsible for action selection, as represented in Figure 1(a). Each module has its own Q matrix representing its partial knowledge of the world



(a)



(b)

Figure 1: Examples of Structural Reinforcement Learning for the classical taxi driver problem: (a) MRL, (b) HRL

state s_i. An agent-level module selection or action selection policy chooses an action from modules preferences, such as the Greatest Mass (GM) strategy [75, 49]:

$$\pi(s_i) = \arg\max_{a \in A} \left\{ \sum_{i=1}^{m} Q_i(s_i, a) \right\}, \tag{3}$$

which selects the action that maximizes the sum of local agent Q-values. The work [30] gives a full review of different action selection policies $\pi : \{A, \mathbb{R}\}^m \rightarrow A$, based on letting the i^{th} agent to propose a single action $a_i \in A$ with an associated weight $w_i \epsilon \mathbb{R}$. In other words, the agent specifies what the a module *"wants to do"* and *"how important"* this action is for it. A variety of different importance interpretations and action selection algorithms are discussed, such as *Minimize Worst Unhappiness*, *Strict Highest W*, *Maximize Best Happiness*, *Maximize Collective Happiness*, and so on. More sophisticated approaches[68, 56] use *gating signals* to decide which module is designed responsible in each state and some authors have studied how to share the reward among modules[77]. The main advantage of this approach is that it allows to learn different concurrent subtasks in a fairly simple way. On the other hand, agent-level action selection could lead to unpredictable behavior and modules may even compete imposing their preferences to the rest.

4.4.2 Hierarchical Reinforcement Learning

Whereas MRL deals with concurrent tasks, HRL decomposes complex tasks into sequentially executed simpler subtasks which are executed from the upper subtask in a recursive manner. Figure 1(b) shows one such a hierarchical decomposition for the taxi problem. Subsystems performing these subtasks are to be separately trained in order to solve the global task.

MAXQ algorithm. One of the most extensively used HRL algorithms is *MAXQ* [18, 19], which is based on HAMQ algorithm [51, 52] and decomposes a MDP into a set of subtasks or subroutines $\{M_0, M_1, \ldots, M_m\}$, each defined as a tuple $M_i = \left\{T_i, A_i, \bar{R}_i\right\}$, where T_i is a subset of states $T_i \in S$ in which subtask M_i is terminated, A_i is a set of allowed actions during execution of M_i (either primitive or composite), and $\bar{R}_i(s')$ is a pseudo-reward function that maps termination states $s' \in T_i$ into real values indicating how desirable they are. The key feature of this approach is that the i^{th} task's value function $Q(i, s, a)$ can be decomposed into two components, nam ely, the expected reward received from executing subtask a, denoted $V(a, s)$, and the expected received reward from the end of subtask a until the completion of parent task M_i, which is also know as the *completion function*:

$$C^{\pi}(i, s, a) = \sum_{s', N} P_i^{\pi}(s', N \mid s, a) \gamma^N Q^{\pi}(i, s', \pi(s')), \tag{4}$$

where $P_i^\pi(s', N \mid s, a)$ represents the probability of observing state s' exactly N time-steps after executing action a in state s following policy π. Then, the function Q^π is expressed as follows

$$Q^\pi(i, s, a) = V^\pi(a, s) + C^\pi(i, s, a), \tag{5}$$

and $V^\pi(i, s)$ is of the form:

$$V^\pi(i, s) = \begin{cases} Q^\pi(i, s, \pi_i(s)) & if\ Composite\ (M_i) \\ \sum_{s'} P(s' \mid s, i)\, R(s' \mid s, i) & if\ Primitive\ (M_i) \end{cases}. \tag{6}$$

This decomposition allows a compact representation and only requires to store the Q function for composite subtasks and V values for primitive actions. Furthermore, MAXQ gives the ability to use different state abstractions because probably not all state variables are relevant to all subtasks. Shared subtasks is another added advantage: for example, subtasks *Reach-Passenger* and *Reach-Goal* in the Figure 1(b) could be collapsed into a single parametrized subtask *Reach(t)* shared by both *Take-Passenger* and *Leave-Passenger*, where t represents the destination as a parameter of the subtask.

This approach relies heavily on the designer's knowledge of the domain and ability to select appropriate subtasks. Because the hierarchy of tasks imposes a hierarchy of policies, each subtasks will reach a locally optimal policy, not taking into account the context in which it is executed, maybe leading to a globally suboptimal policy.

Multi-Agent MAXQ. Although the MAXQ framework was developed for single agent systems, [44] adopted it and extended it to the *Cooperative HRL* algorithm, studying the use of joint-actions to coordinate homogeneous agents. These joint-actions are high-level subtasks (ideally from the level below the root) and thus provide a higher capability to scale up than sharing primitive actions. Agents only have knowledge of what other agents are doing at a high-level (i.e. in a multi-agent taxi scenario, an agent would know whether other agents are approaching a passenger, but not what low-level actions they are performing). This approach implicitly assumes that agents do not interfere with each other and it also implies immediate and reliable communications. A more general algorithm know as *COM-Cooperative HRL* that considered costly but immediate communications and modeled these as an abstraction level below the root node, so each agent learns when to and even with whom to communicate, is presented in [26]. The main drawback remains the dependency upon a correct hand-made design, which is not likely to scale up properly in complex environments, where a more automated approach is more desirable.

4.4.3 Hybrid Structures

Trying to have both the concurrent computation of modular structures and the task decomposition of hierarchical structures, [55] proposes concurrent options

using disjoint action spaces which don't interfere with each other. Another hybrid approach is to define hierarchies of module groups [69], each group responsible of solving a specific subtask in the hierarchy. A big problem that affects nearly all modular approaches and has yet not been solved are the interferences between different modules, which are likely to happen unless they operate on specific disjoint spaces.

4.5 Cooperative Multi-Agent Reinforcement Algorithms

In fully-cooperative systems, agents should coordinate to achieve the team goal and most authors consider a unique shared reward signal. The amount of information shared between agents has been reported to influence the cooperation of a team [70], and three different categories have been proposed according on the degree of coordination of the algorithm: coordination-free, indirectly coordinated and coordinated methods. We will review the most generally applicable methods and refer the interested reader to [11] for a more in-depth review.

4.5.1 Coordination-free MARL

Coordination free MARL can be appropriate for some specific MCRS with tasks such as the displacement of objects by independent mobile robots. In such cases, the coordination is based on the actual state of the object in the environment, which acts as an external independent marker.

The most simple of cooperative MARL methods is to use agents unaware of the actions taken by the rest. This approach offers a great reduction of the state-action space size. It grows linearly instead of the exponential growth (n tables with $\mid S \times A \mid$ entries instead of one with $\mid S \times A^n \mid$ entries) and involves no coordination mechanism. *Independent Learners* were studied in [16] mostly as a benchmark for coordinated methods. Because of the lack of a coordination mechanism, the system cannot be guaranteed to converge to neither a stable nor a globally optimal policy. Nevertheless, it has been quite successfully applied to some cooperative problems with low cooperation requirements.

The work in [41] studied making optimistic assumptions about the behavior of the rest of agents. This method too is only aware of the local action instead of the global joint action, but it updates the Q-matrices only when the resultant state-action value is higher than the previous one. This method has been proved to converge to a globally optimal policy and requires no explicit communication, but is only suitable to deterministic environments.

4.5.2 Indirectly Coordinated MARL

In indirect coordination MARL, the agents try to estimate the policy of the remaining agents, in order to integrate them into the local decision making process. This is the case in some mobile robot applications, such as exploration or robot formation[54]. Some authors have proposed several heuristic algorithms [16, 37] to estimate the most likely response of the rest of agents using models.

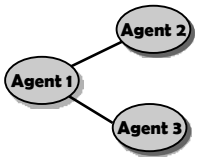


Figure 2: Coordination Graph example

Those models are dynamically built from observed experience and are used to bias local policies towards coordinated joint actions.

On the other hand, each state in a MDP can be regarded as virtual stateless Stochastic Games (SG) and some adaptive methods [47, 73] have been proposed to provably bias local action selection towards a globally optimal joint action. Still, these approaches require additional memory resources and knowledge about the optimal Q^*function, limiting their scalability to increasingly greater problems.

4.5.3 Coordinated MARL

The coordinated MARL perform the integration of the local policies into global (optimal) policies encompassing all the agents. They aim to decide the (optimal) joint-action of all the agents to be performed at each time instant. These approaches can be useful for linked systems, such as the Linked MCRS discussed in [23]. *Distributed Rewards* and *Distributed Value Functions* were studied in [57] as a way to estimulate and control cooperation between neighbors. Instead of updating the state-action values using only the local reward or value functions, agents also used weighted rewards or state-action values of their teammates. This methods scales linearly to the number of agents, but offer no guarantees of optimality.

The *Coordinated Reinforcement Learning* (Coordinated-RL)[29] approximates the global joint value function as a linear combination of local value functions [28]. The complexity of agreeing on a globally optimal joint action can be reduced assuming that agents need not to coordinate with all the rest of agents, but with a smaller subset. These coordination dependencies between agents are context-specific, can change dynamically and can be defined as a *Coordination-Graph* denoted $CG = \{V, E\}$, where undirected edges $e_{ij} \in E$ represent a coordination dependence between agents i and j, such as the one in Figure 2. Each of the n agents has a local Q_i function approximating its contribution to the global function $Q = \sum_{i=1}^{n} Q_i(s_i, a_i)$, and the goal is to select a joint-action $\{a_1, a_2, \ldots, a_n\}$ that maximizes the expected global reward. The use of the CG reduces the state-action space by defining which actions are relevant to each Q_i function, and it can still be further reduced by identifying which state variables are relevant to each local value function ($s_i \in S_i \subseteq S$). In the example of Fig-

ure 2, the coordination task is to find the joint actions that maximize the joint reward given by the addition of the individual rewards:

$$(a_1^*, a_2^*, a_3^*) = \arg \max_{(a_1,a_2,a_3)} \{Q_1(s_1, a_1, a_2, a_3) + Q_2(s_2, a_1, a_2) + Q_3(s_3, a_1, a_3)\}. \tag{7}$$

A *Variable Elimination* (VE) procedure is needed for the agents to agree on a joint-action. The use of a CG gives the chance to maximize the global value function by maximizing one variable at a time, which can be viewed as conditioned maximization. An agent is chosen to communicate its expected local reward for each action to one of its neighbors. Then, this agent can be eliminated from the graph and the selected neighbor can compute that action that maximizes its local value function for each of the possible choices of the first one. 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. While this algorithm can be implemented using a simple message-based protocol and always computes the global optimal joint-action no matter the selected elimination order, time constraints can render this approach not suitable for real-time systems.

Two alternative anytime algorithms were proposed in [72] to agree on a joint-action: *Coordinate Ascent* (CA) and the *Max-Plus* algorithm. Instead of trying to find the absolute maximum, they are both real-time (suboptimal) approximations. CA starts with a randomly generated joint-action and agents change their local action a_i, one at a time, so as to maximize their local function until the global value function cannot be further improved. Depending on time constraints, more than one random start could be generated, and when the time limit is reached, the joint-action with a highest value can be selected. More sophisticated search algorithms, such as evolutionary algorithms, could be used. *Max-Plus* is well-known asynchronous method for estimating the *maximum-a-posteriori* configuration in an undirected graph. Only local messages between agents representing the local value function are needed to compute the globally optimal joint-action, but although it is know to converge in a finite number of steps for graphs without cycles, no guarantees are given about the amount of steps required to converge. A variant of the Max-Plus algorithm is proposed in [39]: agents compute from time to time the global value function and only update the joint-action when updates improve the global value function. A deadline signal is assumed to end the joint-action selection process.

Analogously, *Sparse Cooperative Q-Learning*[38] (or SparseQ) is a distributed version of the traditional Q-Learning for which two global value function decomposition methods have been proposed[39]: agent-based (equivalent to Coordinated-RL) and edge-based. Edge-based decomposition approximates global value for each edge of the CG instead of doing so for each agent: $Q_i = \frac{1}{2} \sum_{e_{ij} \in E} Q_{ij}(s_{ij}, a_i, a_j)$. Two different update rules are given: edge-based and agent-based update. Empirical experiments show that both storage requirements and joint-action calculation for both Coordinated-RL and agent-based SparseQ grow exponentially in the average degree of the CG. Better scalability properties are shown for edge-

based SparseQ when a anytime algorithm is used to approximate the value-maximizing joint-action.

4.6 Transfer Learning

Based on the idea of incremental learning, Transfer Learning (TL) [53, 71] speeds up learning of a *target task* using available knowledge from a *source task*. In a RL context, the general TL approach can be viewed as using knowledge about a *source MDP* $< S', A', P', R' >$ to improve learning in a *target MDP* $< S, A, P, R >$. Several different approaches can be found in the literature. [71] categorizes them by their features: differences allowed between source and target MDPs, how source tasks are selected, how to map different state spaces, the transferred knowledge, the allowed learners and the metrics used to measure improvement.

5 Conclusions and discussion

We propose the MARL as an appropriate paradigm to develop control algorithms for MCRS, describing some of the common issues found applying the basic RL algorithms regarding the desired properties of MCRS presented in Section 1. We have also reviewed some of the main current RL and MARL innovation trends that can lead to scalable solution for MCRS control. In this final section, we will discuss the potential contributions of MARL to review the problems described in Section 3, pointing out which techniques have addressed them and which ones still remain open.

- *Resource scalability:* Two different resources have been considered, memory size and communication bandwidth. The memory requirements can be effectively reduced using any of the existing VFA, because they dramatically reduce the storage requirements and are able to generalize for unknown inputs. RL and MARL performed using VAF can be less accurate because the underlying interpolation may induce some loss of information. They involve training processes besides the RL which can be very sensitive to environment changes, degrading the performance of the system. Automated state abstraction mechanisms, on the other hand, usually require more memory resources than VFA and depend on the topology of the state space. The literature on the subject usually uses highly structured environments which may not always be the case in real MCRS applications. Some compromise between state representation accuracy and memory requirements might be desirable to obtain higher scalability. Communication bandwidth requirements are higher in MARL methods than in single-agent RL scenarios. They can be minimized when a Coordination Graph is used to determine coordination requirements. This dynamic graph is problem dependent. Thus, the scalability of these approaches depends on the specific coordination requirements of the problem.

- *Action heterogeneity:* Temporal abstraction frameworks seem applicable to heterogeneous robotic systems, because action durations can be modeled decomposing them into time-steps. It must be noted, though, that no application example can be found in the literature where a RL algorithm controls several robotic systems with heterogeneous actions and some more work in this area would be interesting.

- *Decentralized control:* MARL algorithms are the natural approach to achieve decentralized control. There are plenty cooperative MARL algorithms in the literature the most interesting areas of research seem to be coordinated and indirectly coordinated MARL algorithms, because they are able to deal with stochastic environments.

- *Control delays:* No relevant literature can be found regarding this issue in RL algorithms. This is a big issue in coordinated MARL algorithms because, even using anytime algorithms, because the state must be first observed and the system must agree on a joint action before an action can be taken. Thus, there will be inevitably some delay from the time an action finishes until the system takes the next one. Using some estimator to predict the next state, such as Kalman Filters, could alleviate this. Assuming the action selection algorithm is executed each T ms, that it needs t_c ms to execute the whole control algorithm and t_p ms to predict the next state, the system could take an action at $t = t_0$, use the state predictor at $t_1 = t_0 + T - t_p - T_c$ and run the entire control algorithm at $t_2 = t_1 + t_c$ using the estimated state instead of the observed one. This way, the system could take next action exactly at $t_3 = t_0 + T$.

- *Robustness to incomplete and noisy sensor data:* The use of PODMP can yield higher applicability than MDP in environments in which not all state variables can be observed, but there is currently no scalable model-free RL algorithm able to deal with them.. Noisy sensor data is a huge problem in coordination-free MARL algorithms, because they all rely on an accurate shared perception of the environment, but in real applications, local measurements are likely to produce different perceptions of the state. An interesting approach could be to use of other AI tools such as Neural Networks to learn the model and estimate non-observable state variables. Using an estimator could also be helpful in the presence of noisy measurements. Communication has also been pointed out as a way of reducing that dependence upon a consistent perception of the environment [11].

- *Convergence time:* Learning the control algorithm in a real environment can be time-wise unaffordable and this is a big issue towards scalable systems. Two different main trends have been found in the literature: transfer learning and task decomposition techniques. Transfer learning can be used to take benefit of simulated off-line experience in a real environment. An interesting line of work would be to simulate off-line models sampled from on-line experience, because this would not require the system designer to have expertise on the specific domain. On the other hand,

task decomposition can be used to reduce the complexity of the task to be solved and therefore reduce the time needed to develop a control algorithm. Manual approaches such as the original MAXQ algorithm require domain knowledge, and we find automated state abstraction and sub-goal identification to be the most interesting topics in this area because automatic complexity reduction approaches are more likely to be scalable. Nevertheless, the literature lacks automatic state abstraction and sub-goal identification examples of MARL applications.

Although it might seem that most of the reviewed issues can be already addressed, there exists no universal scalable solution and more scientific effort should be put towards developing more general methods offering all of the desired MCRS properties. Hybrid approaches mixing both RL methods and other AI tools pose an interesting venue for improving scalability. For example, while model-free learning is very appealing because of its huge applicability to unknown environments, the use of models can alleviate some control issues such as accuracy. An interesting compromise between both worlds could be using model-free learning method to learn a experience-based model, which could be used to predict unknown behaviors.

References

[1] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. In *Machine Learning*, pages 37–66, 1991.

[2] R. Aragues, J. Cortes, and C. Sagues. Distributed consensus algorithms for merging feature-based maps with limited communication. *Robotics and Autonomous Systems*, 59(3-4):163 – 180, 2011.

[3] Andrew G. Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *In Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758. ACM Press, 2004.

[4] Hamid Berenji. Fuzzy reinforcement learning and dynamic programming. In Anca Ralescu, editor, *Fuzzy Logic in Artificial Intelligence*, volume 847 of *Lecture Notes in Computer Science*, pages 1–9. Springer Berlin / Heidelberg, 1994.

[5] H.R. Berenji. Fuzzy Q-learning for generalization of reinforcement learning. In IEEE Press, editor, *Proc. of the Fifth IEEE International Conference on Fuzzy Systems*, volume 3, pages 2208 – 2214, 1996.

[6] Daniel S. Bernstein. Dynamic programming for partially observable stochastic games. In *In Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 709–715, 2004.

[7] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. In *Mathematics of Operations Research*, page 2002, 2000.

[8] Michael Bowling and Manuela Veloso. Scalable learning in stochastic games. In *In: AAAI Workshop on Game Theoretic and Decision Theoretic Agents*, pages 11–18, 2002.

[9] Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, pages 369–376. MIT Press, 1995.

[10] Steven J. Bradtke and Michael O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Advances in Neural Information Processing Systems*, pages 393–400. MIT Press, 1994.

[11] L. Busoniu, R. Babuska, and B. De Schutter. Comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, 38(2):pp. 156–172, 2008.

[12] D. Chapman and L.P. Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Learning and Knowledge Acquisition, IJCAI 1991*, pages 726–731. Morgan Kaufmann, 1991.

[13] Chung-Cheng Chiu and Von-Wun Soo. Subgoal identification for reinforcement learning and planning in multiagent problem solving. In Paolo Petta, Jï¿œrg Mï¿œller, Matthias Klusch, and Michael Georgeff, editors, *Multiagent System Technologies*, volume 4687 of *Lecture Notes in Computer Science*, pages pp. 37–48. Springer Berlin / Heidelberg, 2007.

[14] Chung-Cheng Chiu and Von-Wun Soo. Automatic complexity reduction in reinforcement learning. *Computational Intelligence*, 26(1):pp. 1–25, 2010.

[15] Chung-Cheng Chiu and Von-Wun Soo. *Advances in Reinforcement Learning*, chapter Subgoal Identifications in Reinforcement Learning: A Survey, pages pp.181–188. InTech, 2011.

[16] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752. AAAI Press, 1997.

[17] Robert Crites and Andrew Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*, pages 1017–1023. MIT Press, 1996.

[18] Thomas Dietterich. An overview of maxq hierarchical reinforcement learning. In Berthe Choueiry and Toby Walsh, editors, *Abstraction, Reformulation, and Approximation*, volume 1864 of *Lecture Notes in Computer Science*, pages pp. 26–44. Springer Berlin / Heidelberg, 2000.

[19] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:pp. 227–303, 2000.

[20] Bruce Digney. Learning hierarchical control structures for multiple tasks and changing environments. In *In Proceedings of the Fifth Conference on the Simulation of Adaptive Behavior: SAB 98*. MIT Press, 1998.

[21] Y. Duan and X. Hexu. Fuzzy reinforcement learning and its application in robot navigation. In *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, volume 2, pages 899 –904 Vol. 2, 18-21 2005.

[22] R.J. Duro, Manuel Graña, and J. de Lope. On the potential contributions of hybrid intelligent approaches to multicomponen robotic system development. *Information Sciences*, 180(14):2635–2648, 2010.

[23] Z. Echegoyen, I. Villaverde, R. Moreno, M. Graña, and A. d'Anjou. Linked multi-component mobile robots: modeling, simulation and control. *Robotics and Autonomous Systems*, 58(12):1292–1305, 2010.

[24] B. Fernandez-Gauna, J.M. Lopez-Guede, E. Zulueta, Z. Echegoyen, and M. Graña. Basic results and experiments on robotic multi-agent system for hose deployment and transportation. *International Journal of Artificial Intelligence*, 6(S11):183–202, 2011.

[25] Robert Fitch, Bernhard Hengst, Dorian Suc, Greg Calbert, and Jason Scholz. Structural abstraction experiments in reinforcement learning. In Shichao Zhang and Ray Jarvis, editors, *AI 2005: Advances in Artificial Intelligence*, volume 3809 of *Lecture Notes in Computer Science*, pages pp. 164–175. Springer Berlin / Heidelberg, 2005.

[26] Mohammad Ghavamzadeh and Sridhar Mahadevan. Learning to communicate and act using hierarchical reinforcement learning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, pages 1114–1121, 2004.

[27] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in markov decision processes. *Artif. Intell.*, 147:163–223, July 2003.

[28] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored mdps. In *NIPS-14*, pages pp. 1523–1530. The MIT Press, 2001.

[29] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *In Proceedings of the IXth ICML*, pages 227–234, 2002.

[30] T. Hall, M. Humphrys, and M. Humphrys. Action selection methods using reinforcement learning. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 135–144. MIT Press, 1996.

[31] Bernhard Hengst. Discovering hierarchy in reinforcement learning with hexq. In *In Maching Learning: Proceedings of the Nineteenth International Conference on Machine Learning*, pages pp. 243–250. Morgan Kaufmann, 2002.

[32] Pieter Hoen, Karl Tuyls, Liviu Panait, Sean Luke, and J.A. La Poutrï¿œ. An overview of cooperative and competitive multiagent learning. In Karl Tuyls, Pieter Hoen, Katja Verbeeck, and Sandip Sen, editors, *Learning and Adaption in Multi-Agent Systems*, volume 3898 of *Lecture Notes in Computer Science*, pages 1–46. Springer Berlin / Heidelberg, 2006.

[33] Nicholas K. Jong. State abstraction discovery from irrelevant state variables. In *In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages pp. 752–757, 2005.

[34] Anders Jonsson and Andrew Barto. A causal approach to hierarchical decomposition of factored mdps. In *Advances in Neural Information Processing Systems*, volume 13, pages pp.1054–1060, 2005.

[35] Anders Jonsson and Andrew Barto. Causal graph based decomposition of factored mdps. *J. Mach. Learn. Res.*, 7:pp. 2259–2301, December 2006.

[36] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[37] S. Kapetanakis and D. Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. In *18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence*, pages pp. 326–331, 2002.

[38] Jelle R. Kok and Nikos Vlassis. Sparse cooperative q-learning. In *Proceedings of the International Conference on Machine Learning*, pages 481–488. ACM, 2004.

[39] Jelle R. Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7:1789–1828, 2006.

[40] Daphne Koller and Ronald Parr. Computing factored value functions for policies in structured mdps. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1332–1339. Morgan Kaufmann, 1999.

[41] Martin Lauer and Martin A. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 535–542, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[42] C. Li, J. Zhang, and Y. Li. Application of artificial neural network based on q-learning for mobile robot path planning. In *Information Acquisition, 2006 IEEE International Conference on*, pages 978 –982, 20-23 2006.

[43] Sridhar Mahadevan, Nicholas Marchalleck, Tapas K. Das, and A. Gosavi. Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *Proceedings of the 14th International Conference on Machine Learning*, pages 202–210. Morgan Kaufmann, 1997.

[44] Rajbala Makar and Sridhar Mahadevan. Hierarchical multi-agent reinforcement learning. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages pp. 246–253. ACM Press, 2001.

[45] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *In Proceedings of the Twenty-First International Conference on Machine Learning*, pages pp. 560–567. ACM Press, 2004.

[46] Amy Mcgovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *In Proceedings of the eighteenth international conference on machine learning*, pages pp. 361–368. Morgan Kaufmann, 2001.

[47] Francisco Melo and M. Ribeiro. Coordinated learning in multiagent mdps with infinite state-space. *Autonomous Agents and Multi-Agent Systems*, 21:321–367, 2010. 10.1007/s10458-009-9104-y.

[48] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut: Dynamic discovery of sub-goals in reinforcement learning. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Machine Learning: ECML 2002*, volume 2430 of *Lecture Notes in Computer Science*, pages pp. 187–195. Springer Berlin / Heidelberg, 2002.

[49] N. Ono and K. Fukumoto. A modular approach to multi-agent reinforcement learning. In Gerhard Weiss, editor, *Distributed Artificial Intelligence Meets Machine Learning Learning in Multi-Agent Environments*, volume 1221 of *Lecture Notes in Computer Science*, pages 25–39. Springer Berlin / Heidelberg, 1997.

[50] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[51] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, pages pp. 1043–1049. MIT Press, 1998.

[52] Ronald Edward Parr. Hierarchical control and learning for markov decision processes. Master's thesis, University of California, Berkeley, 1998. AAI9902197.

[53] Marc Ponsen, Matthew E. Taylor, and Karl Tuyls. Abstraction and generalization in reinforcement learning: A summary and framework. In *ALA Workshop, Adaptive and Learning Agents (LNAI Journal)*, pages pp. 1–33, 2010.

[54] Wei Ren and R.W. Beard. *Distributed Consensus in Multi-Vehicle Cooperative Control: Theory and Applications*. Springer Publishing Company, Incorporated, 2007.

[55] Khashayar Rohanimanesh and Sridhar Mahadevan. Decision-theoretic planning with concurrent temporally extended actions. In *In UAI'01*, pages pp. 472–479. Morgan Kaufmann Publishers, 2001.

[56] Kazuyuki Samejima, Kenji Doya, and Mitsuo Kawato. Inter-module credit assignment in modular reinforcement learning. *Neural Netw.*, 16:985–994, September 2003.

[57] Jeff Schneider, Weng-Keen Wong, Andrew Moore, and Martin Riedmiller. Distributed value functions. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 371–378. Morgan Kaufmann, 1999.

[58] Anton Maximilian Schï¿œfer, Steffen Udluft, and Departement Neural Computation. Solving partially observable reinforcement learning problems with recurrent neural networks. In *In Workshop Proc. of the European Conference on Machine Learning*, 2005.

[59] Jing Shen, Guochang Gu, and Haibo Liu. Multi-agent hierarchical reinforcement learning by integrating options into maxq. In *Computer and Computational Sciences, 2006. IMSCCS '06. First International Multi-Symposiums on*, volume 1, pages 676–682, 2006.

[60] Ozgur Simsek, Alicia P. Wolfe, and Andrew G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *In Proceedings of the Twenty-Second International Conference on Machine Learning*, pages pp. 816–823, 2005.

[61] Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesv Ari. Convergence results for single-step on-policy reinforcement-learning algorithms. In *Machine Learning*, pages 287–308, 1998.

[62] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems 7*, pages 361–368. MIT Press, 1995.

[63] William D. Smart. Explicit manifold representations for value-function approximation in reinforcement learning. In *Prceedings of the 8th International Symposium on Artificial Intelligence and mathematics*, pages 25–2004, 2004.

[64] Richard Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:pp. 181–211, 1999.

[65] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

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

[67] Prasad Tadepalli and Dokyeong Ok. Scaling up average reward reinforcement learning by approximating the domain models and the value function. In *In Saitta*, pages 471–479. Morgan Kaufmann, 1996.

[68] Y. Takahashi and M. Asada. *Reinforcement Learning: Theory and Applications*, chapter Modular Learning Systems for Behavior Acquisition in Multi-Agent Environment, pages 225–238. I-Tech Education and Publishing, Vienna, 2008.

[69] Yasutake Takahashi and Minoru Asada. Modular learning systems for soccer robot. In *Proceedings of the Fourth International Symposium on Human and Artificial Intelligence Systems*, pages pp.370–375, 2004.

[70] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann, 1993.

[71] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[72] N. Vlassis, R. Elhorst, and J. R. Kok. Anytime algorithms for multiagent decision making using coordination graphs. In *In Proc. Intl. Conf. on Systems, Man and Cybernetics*, 2004.

[73] Xiaofeng Wang and Tuomas Sandholm. Reinforcement learning to play an optimal nash equilibrium in team markov games. In *in Advances in Neural Information Processing Systems*, pages 1571–1578. MIT Press, 2002.

[74] Christopher Watkins and Peter Dayan. Technical note: Q-learning. In *Machine Learning*, volume 8, pages pp. 279–292, May 1992.

[75] S. Whitehead, J. Karlsson, and J. Tenenberg. *Robot Learning*, chapter Learning multiple goal behavior via task decomposition and dynamic policy merging, pages 45–78. Kluwer Academic Publisher, 1993.

[76] H. Xiao, L. Liao, and F. Zhou. Mobile robot path planning based on q-ann. In *Automation and Logistics, 2007 IEEE International Conference on*, pages 2650 –2654, 18-21 2007.

[77] Pucheng Zhou and Bingrong Hong. A modular on-line profit sharing approach in multiagent domains. *International Journal of Electrical and Computer Engineering*, 1(6):424–431, 2006.