Modular Multi-Agent Reinforcement Learning of Linked Multi-Component Robotic Systems

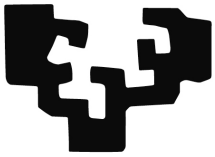
By

Borja Fernández Gauna

http://www.ehu.es/ccwintco

Dissertation presented to the Department of Computer Science and Artificial Intelligence in partial fulfillment of the requirements for the degree of

Doctor of Philosophy



PhD Adviser:

Prof. Manuel Graña Romay

At

The University of the Basque Country
Donostia - San Sebastián
2012

Universidad del País Vasco Euskal Herriko Unibertsitatea

AUTORIZACION DEL/LA DIRECTOR/A DE TESIS

PARA SU PRESENTACION

Dr/a. _____con N.I.F._____

como Director/a de la Tesis Doctoral: _____

_____

_____

realizada en el Departamento _____

_____

por el Doctorando Don/ña. _____ ,

autorizo la presentación de la citada Tesis Doctoral, dado que reúne las condiciones necesarias para su defensa.

En _____a _____de _____de _____

EL/LA DIRECTOR/A DE LA TESIS

Fdo.: _____

Universidad del País Vasco Euskal Herriko Unibertsitatea

CONFORMIDAD DEL DEPARTAMENTO

El Consejo del Departamento de _____

_____

en reunión celebrada el día ____ de _____ de _____ ha acordado dar la conformidad a la admisión a trámite de presentación de la Tesis Doctoral titulada: _____

_____

_____

dirigida por el/la Dr/a. _____

_____

y presentada por Don/ña. _____

ante este Departamento.

En _____a _____de _____de _____

Vº Bº DIRECTOR/A DEL DEPARTAMENTO SECRETARIO/A DEL DEPARTAMENTO

Fdo.: _____ Fdo.: _____

Universidad del País Vasco Euskal Herriko Unibertsitatea

ACTA DE GRADO DE DOCTOR

ACTA DE DEFENSA DE TESIS DOCTORAL

DOCTORANDO DON/ÑA. _____

TITULO DE LA TESIS: _____

_____

_____

El Tribunal designado por la Subcomisión de Doctorado de la UPV/EHU para calificar la Tesis Doctoral arriba indicada y reunido en el día de la fecha, una vez efectuada la defensa por el doctorando y contestadas las objeciones y/o sugerencias que se le han formulado, ha otorgado por_____la calificación de:

unanimidad ó mayoría

Idioma/s defensa: _____

En _____a _____de _____de _____

EL/LA PRESIDENTE/A, EL/LA SECRETARIO/A,

Fdo.: Fdo.:

Dr/a: _____ Dr/a: _____

VOCAL 1º, VOCAL 2º, VOCAL 3º,

Fdo.: Fdo.: Fdo.:

Dr/a: _____Dr/a:_____Dr/a:_____

EL/LA DOCTORANDO/A,

Fdo.: _____

Acknowledgments

I would like to express my gratitude to Prof. Manuel Graña, my PhD thesis adviser, for guiding my exploration in this work's reinforcement learning process. He helped me focus on the relevant state variables, and without his help, I would not have been able to reach the goal.

I also want to thank the rest of the members of the Computational Intelligence research group for their cooperation.

Finally, I want to thank my family for initializing my Q-values and for their unconditional love. Especially, I would like to thank my parents Rosa and Javier and my brothers Igor, Ivan and Julen.

Thank you very much.

Borja Fernández Gauna

Modular Multi-Agent Reinforcement Learning of Linked Multi-Component Robotic Systems

by

Borja Fernández Gauna

Submitted to the Department of Computer Science and Artificial Intelligence on March 16, 2012,
in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

The contents of this Thesis can be summarized as two main ideas: modular techniques to decompose a reinforcement learning task in over-constrained environments such as Linked-MCRS as several concurrent sub-tasks, and extension of these modular reinforcement learning approaches to multi-agent reinforcement learning environments.

Modular decomposition of a over-constrained reinforcement learning task allows agents to learn each decomposed reward signal using only the relevant subset of state variables. The straightforward benefit of this approach is a reduction of the learning problem dimensionality. We propose in this work a specific modular structure, in which one module learns the main task while several other modules concurrently learn how to avoid taking state-actions known to lead to undesired states (i.e. broken constraints). To this end, we have studied two different approaches: thresholded state-action vetoes and safe state-action vetoes. Computational experiments show that the modular decomposition leads to an efficient learning of undesired transitions that boosts the learning speed. In order to reduce the unaffordable computational time needed to learn in an accurate simulation environment, we have studied training agents first in a Partially Constrained Models (PCM), which is an under-constrained and computationally less-demanding environment. Both theoretical analysis and empirical tests confirm that the time requirements are greatly reduced using PCM. We have also studied the prediction of undesired transitions using existing two-class classifiers, which aid the reinforcement learner predicting the desirability of each eligible action.

Successful control of Linked-MCRS requires decentralized control algorithms. We have extended modular and classifier-based approaches to the multi-agent

case using Round-Robin scheduling action execution, in which agents learn the value of local actions in turns, thus reducing the communication and storage requirements, and assuring a stationary environment during each agent's turn. We propose a message-based procedure to approximate the optimal joint-policy using the local values. This non-concurrent learning scheme presents some advantages if compared with Distributed Q-Learning: it does not require a deterministic environment to work properly, and agents can learn the desirability of local actions without interferences from other agents.

Our work-bench in this work is an accurate but computationally very expensive hose transportation task that uses Geometrically Exact Dynamic Splines as the hose model. We have also used a computationally less-expensive simplified version of the hose transportation task for some of our experiments. Finally, aiming at a broader validation of our techniques, we have also conducted experiments on the classical multi-agent taxi problem.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

In this introductory chapter, we give the context of this Thesis. First, we provide the motivation for this work in Section 1.1 and list its objectives in Section 1.2. Section 1.3 highlights the main contributions made in this Thesis, and Section 1.3.1 enumerates the most relevant publications. Section 1.4 comments the structure of this dissertation and, finally, Section 1.5 reviews some of the notation used throughout this Thesis.

1.1 Motivation

There are several real-life tasks which involve the manipulation of non-rigid physical elements, such as hoses or wires, as a way to transport fluids or energy. Some examples of hose manipulating tasks would be: fuel supplying, firefighting or cleaning. Figures 1.1a and 1.1b show two instances: a group of women and men cleaning the inside of a shipyard, and a group of firefighters carrying a hose through a narrow corridor to fight a fire inside some industrial facilities. These kind of hose manipulation tasks may even imply hazardous conditions. Depending on the fluid pressure and size of the hose, they may require a great deal of manpower to control. Therefore, for safety reasons, unmanned robotic systems are a preferable solution in industrial or hazardous environments.

Linked Multi-Component Robotic Systems (L-MCRS) are defined as sets of mobile robotic units physically linked by a non-rigid element, such as a hose. Classical robotic control approaches have usually aimed at producing pre-programmed behaviors that make the system able to complete the task, but these kind of methods are hardly able to adapt to noisy and changing environments. Reinforcement learning algorithms (RL) seem to be a more adequate approach for L-MCRS systems, because:

- agents require little human interaction to learn the task,
- they can produce decentralized and distributed control algorithms,
- they can adapt to changing environments,

- they can save a lot of development time if first trained on a computer and then implemented in the real robotic units.

Among the existing RL algorithms, we have adopted Q-learning as the benchmark because of its two main properties:

- through repeated interaction with the environment, a Q-learning agent is guaranteed to learn an optimal policy, and
- the agent does not require knowledge about the underlying dynamic model.

Modeling the non-rigid links is a non-trivial issue, but one which is critical in order to be able to study those systems either analytically or via simulation. Some dynamic modeling techniques for non-rigid uni-dimensional objects are reviewed in [42, 34]: differential equations[87], rigid element chains[59], spring mass systems[50], combining spline geometrical models and physical dynamical models[92], and spline models combined with the Cosserat rod theory [116], also known as Geometrically Exact Dynamic Splines (GEDS). In this work, we have adopted the Geometrically Exact Dynamic Splines (GEDS) (Appendix A), which we regard as the state-of-the-art for nearly uni-dimensional non-rigid elements. This is a very accurate but also computationally very expensive model. It also includes physical constraints, which force the simulation environment to restart the learning episodes if broken. Because Q-learning requires a large amount of interaction with the environment for the agent to learn, this model poses some challenges:

- unaffordable computational learning time,
- dimensionality of the hose configuration's characterization,
- physical constraints.

The task of manipulating the hose is restricted to a particular case: a set of robots are uniformly distributed along the hose and the goal of the system is to control the robots so the tip of the hose reaches a specific goal position. We consider this as an initial approach to more complex tasks which may include more complex scenarios.

1.2 Objectives

The main objective of this Thesis is to propose enhancements to standard Q-Learning, so it can learn an optimal decentralized control algorithm for Linked Multi-Component Robotic Systems. This work restricts the set of scenarios where L-MCRS could be used to a specific instance of hose transportation, which, at this point, is a more realistic goal. The specific objectives are the following:

1. Review the literature on the use of RL to learn an optimal control algorithm for Linked-MCRS and regular Multi-Component Robotic Systems.

(a) Workers directing water hoses for cleaning purposes in a ship.

(b) Firefighters carrying a hose through a narrow corridor.

Figure 1.1: Real-life tasks that involve manipulating a hose.

2. Propose methods to reduce the dimensionality of the learning task without reducing the quality of the control algorithm learnt. For the system to be scalable to bigger MCRS, the main issue to address is dimensionality.

3. Define safe task-exploration policies. In simulated over-constrained scenarios such as L-MCRS, physical constraints can hinder seriously the learning performance if they are not properly managed. In real-life experiments, breaking them may even damage the linking elements or the robotic equipment.

4. Study the use of these algorithms for multi-agent implementations of the hose transportation task. Multi-agent Reinforcement Learning algorithms pose some inherent difficulties which must also be observed, such as communication-requirements and coordination.

5. Investigate whether the proposed Q-learning enhancements are adequate, not only for L-MCRS scenarios, but also for more general multi-agent scenarios.

6. Define a computationally cheaper training environment for the agents to learn the basic skills of the task. We expect these basic skills to help the agent make a more efficient use of time in the goal task.

1.3 Contributions of the Thesis

According to the objective list in the previous section, we now highlight the main contributions of this as follows:

1. A simplified model for the hose transport task has been proposed.

2. Aiming at reducing the dimensionality of multi-agent approaches, a novel distributed framework has been proposed in which agents are not able to observe all the state variables, but only a few. A modification of Q-Learning has also been proposed to learn in such partially observable environments.

3. Modular RL architectures have been proposed so as to decompose the main task into several sub-tasks, each of which may only require a subset of the state variables, thus reducing the state dimensionality.

4. Three different methods have been proposed to learn how not to reach undesired states, such as broken constraints:

   (a) we explore the use of risk thresholds so as to avoid taking dangerous actions,

   (b) we study the use of safe policies with, after some exploration time, reduce the probability of taking risky actions,

(c) supervised learning methods have also been applied so as to approach the constraint learning task in a more general way.

5. The dimensionality of the multi-agent approaches has also been approached using a novel distributed algorithm, in which agents take actions following execution turns and, after learning the Q-values, an optimal joint-policy is approximated using a message-passing scheme.

6. We have studied the applicability of modular RL, undesired states learning and distributed multi-agent algorithms in the classical multi-agent taxi domain.

7. We have theoretically studied the learning improvement introduced by first training the agents on the simplified hose model has theoretically been studied. Also, computational experiments have been conducted to validate the theoretical conclusions.

1.3.1 Publications

- B. Fernandez-Gauna, J. M. Lopez-Guede, and E. Zulueta, *"Linked Multi-Component robotic systems: Basic assessment of linking element dynamical effect"* in Hybrid Artificial Intelligence Systems, ser. Lecture Notes in Computer Science, M. Graña Romay, E. Corchado, and M. Garcia Sebastian, Eds. Springer Berlin / Heidelberg, 2010, vol. 6076, pp. 73-79, ISBN: 978-3-642-13768-6

- J. M. Lopez-Guede, E. Zulueta, B. Fernandez and M. Graña, *"Multi-Robot Systems Control Implementation"* in Robot Learning, Suraiya Jabin (Ed.), pp.137-150. Sciyo 2010. ISBN 978-953-307-104-6

- B. Fernandez-Gauna, J. M. Lopez-Guede, E. Zulueta, and M. Graña, *"Learning hose transport control with q-learning"*, Neural Network World, vol. 20, no. 7, SI, pp. 913-923, 2010, ISSN: 1210-0552

- 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, vol. 6, no. S11, pp. 183-202, March 2011, ISSN: 0974-0635

- M. Graña, B. Fernandez-Gauna, and J. M. Lopez-Guede, *"Cooperative multi-agent reinforcement learning for multi-component robotic systems: guidelines for future research"*, Paladyn, Journal of Behavioral Robotics, pp. 1-11, 2011, 10.2478/s13230-011-0017-5. [Online].
Available: http://dx.doi.org/10.2478/s13230-011-0017-5, ISSN: 2080-9778

- J. M. Lopez-Guede, B. Fernandez-Gauna, M. Graña, and E. Zulueta, *"Empirical study of q-learning based elemental hose transport control"* in

Hybrid Artificial Intelligent Systems, ser. Lecture Notes in Computer Science, E. Corchado, M. Kurzynski, and M. Wozniak, Eds. Springer Berlin / Heidelberg, 2011, vol. 6679, pp. 455-462, ISBN: 978-3-642-21218-5

- B. Fernandez-Gauna, J. M. Lopez-Guede, and M. Graña, *"Towards concurrent q-learning on linked multi-component robotic systems"* in Hybrid Artificial Intelligent Systems, ser. Lecture Notes in Computer Science, E. Corchado, M. Kurzynski, and M. Wozniak, Eds. Springer Berlin / Heidelberg, 2011, vol. 6679, pp. 463-470, ISBN: 978-3-642-21218-5

- B. Fernandez-Gauna, J. M. Lopez-Guede, and M. Graña, *"Concurrent modular q-learning with local rewards on linked multicomponent robotic systems"* in Foundations on Natural and Artificial Computation, ser. Lecture Notes in Computer Science, J. Ferrández, J. Alvarez Sánchez, F. de la Paz, and F. Toledo, Eds. Springer Berlin / Heidelberg, 2011, vol. 6686, pp. 148-155, ISBN: 978-3-642-21343-4

- J. M. Lopez-Guede, B. Fernandez-Gauna, M. Graña, and E. Zulueta. *"Further Results Learning Hose Transport Control with Q-learning"*, JoPhA, Journal of Physical Agents, 2011, ISSN: 1888-0258 (accepted, in press)

- J. M. Lopez-Guede, B. Fernandez-Gauna, M. Graña, and E. Zulueta, *"TRQ-learning: Improving the control of single robot hose transport"*, Cybernetics and Systems, 2011, ISSN: 0196-9722 (accepted, in press)

- J. M. Lopez-Guede, B. Fernandez-Gauna, M. Graña, and E. Zulueta. *"Visual Detection in Linked Multi-Component Robotic System"*, Robotic Vision: Technologies for Machine Learning and Vision Applications. Ed. José García Rodríguez and Miguel Cazorla. IGI Global. 2011 (accepted, in press)

- B. Fernandez-Gauna, J. M. Lopez-Guede, M. Graña. *"Modular Q-Learning with State-Action Vetoes for Linked Multi-component Robotic Systems"*. Internation Journal of Applied Mathematics and Computer Science. (submitted)

- B. Fernandez-Gauna, J. M. Lopez-Guede, M. Graña . *"Transfer Learning with Partially Constrained Models: application to reinforcement learning of linked multicomponent robot system control"*. Robotic and Autonomous Systems. (submitted)

- B. Fernandez-Gauna, J. M. Lopez-Guede, M. Graña. *"A Round-Robin Distributed Q-Learning for Over-constrained Multi-Agent Systems"*. Autonomous Agents and Multi-Agent Systems. (submitted)

1.4 Structure of the dissertation

This dissertation has the following structure.

Chapter 2 contains a review of our research group's first attempts on the hose manipulating tasks and a detailed description of the main problems encountered in this initial phase of work.

Chapter 3 reviews the state of the art regarding the use of RL to learn the control algorithm for MCRS and some ideas for future research.

Chapter 4 proposes a partially observable framework as a dimensionality reduction method for Linked Multi-Component Robotic Systems. Also, we propose a thresholded modular system in order to limit the probability to reach an undesired state.

Chapter 5 explores the use of modular architectures and safe policies in single-agent environments and proposes an algorithm to automatically determine which state variables are relevant for each RL module.

Chapter 6 extends the safe-veto policies to distributed multi-agent environments, in which agents need not be aware of other agents' decisions.

Chapter 7 proposes a new Transfer Learning framework using Partially Constrained Models, and the learning gain is studied from a theoretical viewpoint.

Chapter 8 makes use of existing supervised learning algorithms as a way to learn constraints using such an developed area of knowledge.

Appendix A gives a detailed description of the GEDS model.

Appendix B describes the three different environments used in our simulations: the GEDS hose transportation task, the simplified hose transportation and the multi-agent taxi domain.

1.5 Notation

X	set of all state variables
S	set of all possible states of the state space, defined as the cartesian product of the range of values of the state variables X, such that $S = S^G \cup S^U \cup S^T$
G	subset of states where the goal has been reached, such that $G \subset S$
U	subset of states where a failure or an undesired situation occurs, such that $U \subset S$
T	subset of inconclusive states, such that $T \subset S$ and $T = S - G - U$
A	set of all available actions
$A(s)$	set of all available actions in the state s
P	state transition function, such that $P: S \times A \to S$ or $P: S \times A \times S \to \mathbb{R}$
R	reward function, which is indistinctly defined as $R(s)$ ($R: S \times A \times S \to \mathbb{R}$), $R(s, a, s')$ ($R: S \times A \to \mathbb{R}$) or $R(s)$ ($R: S \to \mathbb{R}$).

$\|S\|$	number of elements of a set S
t	discrete time step
s, s_t	state, usually at time t, such that $s \in S$ and $s_t \in S$
s', s_{t+1}	state, usually at time $t+1$, such that $s' \in S$ and $s_{t+1} \in S$
a, a_t	action, usually at time t, such that $a \in A$ and $a_t \in A$
r, r_t	immediate reward, usually at time t, such that $r \in \mathbb{R}$ and $s \in S$
$V^*(s)$	state value function, value of state $s \in S$ an optimal policy, such that $V^* : S \to \mathbb{R}$
$V^\pi(s)$	state value function, value of state $s \in S$ under a policy π, such that $V^\pi : S \to \mathbb{R}$
V	estimated of a state value function $V^*(s)$ or $V^\pi(s)$, such that $V : S \to \mathbb{R}$
$Q^*(s, a)$	state-action value function, value of taking $a \in A$ in $s \in S$ under an optimal policy, such that $Q^* : S \times A \to \mathbb{R}$
$Q^\pi(s, a)$	state-action value function, value of taking $a \in A$ in $s \in S$ under a policy π, such that $Q^\pi : S \times A \to \mathbb{R}$
$Q(s, a)$	estimates of a state-action value function $Q^*(s, a)$ or $Q^\pi(s, a)$, such that $Q : S \times A \to \mathbb{R}$
π	policy, such that $\pi : S \to A$ or $\pi : S \times A \to \mathbb{R}$
π^*	optimal policy maximizing the state value function or the state-action value function
$\pi_\epsilon(s, a)$	ϵ-greedy policy returning the best action available in state $s \in S$ according to the current estimates Q of the optimal state-action value function Q^*
α	step-size parameter that determines how new and old information are averaged, such that $0 < \alpha \leq 1$
γ	discount rate parameter that determines the importance if future rewards, such that $0 < \gamma \leq 1$
ϵ	probability of choosing a random action in ϵ-greedy policy, such that $0 \leq \epsilon \leq 1$

Chapter 2

On the control of L-MCRS

The main application topic of the Thesis is the control of Linked Multicomponent Robotic Systems (L-MCRS). In this chapter, we introduce this kind of systems, which have been a research line for the Computational Intelligence group in the last years. We comment the main properties required of the control algorithms for such kinds of systems. We introduce the main issues dealt with along the Thesis. One is the efficient exploration of state spaces in the case of over-constrained systems. The other is the application of efficient cooperative Multi-Agent strategies matching the multi-component structure of the systems. Specifically Multi-Agent Reinforcement Learning (MARL), because we are proposing Reinforcement Learning (RL) as the main framework for autonomous learning of the system's control.

The chapter is structured as follows. Section 2.1 gives an introduction to MCRS control algorithms. Section 2.2 gives details on the specific L-MCRS we treat in this Thesis. Section 2.3 comments on the issues raised by over-constrained system. Section 2.4 gives some comments on MARL issues.

2.1 Introduction

According to the Multi-Component Robotic System (MCRS) categorization proposed in [33], Linked MCRS (L-MCRS) are defined as a collection of autonomous robots linked by (or attached to) a non-rigid physical link. They are distinguished from Distributed MCRS (D-MCRS), which are uncoupled groups of robots, and Modular MCRS (M-MCRS), which are rigidly linked modular robots. MCRS 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 [6], establishing dynamic communication links, and driving a hose to a goal [35, 41].

The most salient desired properties of a MCRS control algorithm 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. This is still worse in the case of L-MCRS, where the complexity of the non-linear model is even greater. The study of L-MCRS is a novel research and no relevant information about this subject can be found in the literature, besides our group's own publications. We are currently focused on the accurate modeling of these systems and development of control algorithms. We have started dealing with control and modeling of these systems in [36, 35] and [40]. The latter showed that even a simple spring model of the physical-link in a cooperative control problem introduces highly non-linear dynamics in the system, making it a hard task to derive the control commands. Poor individual performance can be easily propagated to the whole team due to the non-rigid

physical element. The paradigm of L-MCRS is exemplified by the task of carrying a hose from the origin point to a predefined destination using a collection of autonomous robots attached to the hose. Because of the inherent complexity of robot dynamics, further increased by the complex model of the hose, the system is not solvable in an analytic fashion. We focus on Artificial Intelligence techniques, specifically Reinforcement Learning (RL), 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.

2.2 Specific problem definition

Along the Thesis, we will be dealing with a specific paradigmatic example of L-MCRS composed of one hose attached to a fixed end (the source) and whose other end (the tip) is transported by a mobile robot attached to it. For convenience and without lack of generality, we assume that the source is located at the origin of the working space where the hose is deployed. Several other robots may be attached to fixed points along the length of the hose. The task imposed to the team of robots is to deploy the hose so that the tip of the hose reaches a predefined position. This is an instance of cooperative work, because all robots have the same goal. Figure 2.1 exemplifies several configurations of this system for a single robot corresponding to the evolution of an episode of the hose transportation applying a control algorithm learned by single agent RL [38].

The number of the robots scales the complexity of the problem. In this Thesis we are dealing with true multi-component systems, that is, we have more than one robot in all experimental instances. A typical question that we pose is the sensitivity of the learning control algorithm to the increase in scale.

The learning process is carried out on simulations of the system. This allows for fast realization of the learning experiments, reproducibility of the results, complete observation of all the variables involved in the process for verification and validation, and extensive parameter sensitivity tuning, at the cost of losing contact with the physical reality of the system. The complexity of the simulation model is proportional to its accuracy. Modeling the non-rigid links of L-MCRS is a non-trivial issue, but one which is critical in order to be able to study those systems either analytically or via simulation. Some dynamic modeling techniques for non-rigid uni-dimensional objects are reviewed in [36, 35]: differential equations [87], rigid element chains [59], spring mass systems [50], combining spline geometrical models and physical dynamical models [92], and spline models combined with the Cosserat rod theory [116], also known as Geometrically Exact Dynamic Splines (GEDS). Throughout this Thesis, we perform RL on two hose models:

- an accurate hose model based on GEDS whose details are given in Appendix A
- a first order approximation, where the hose segment are modeled as lines

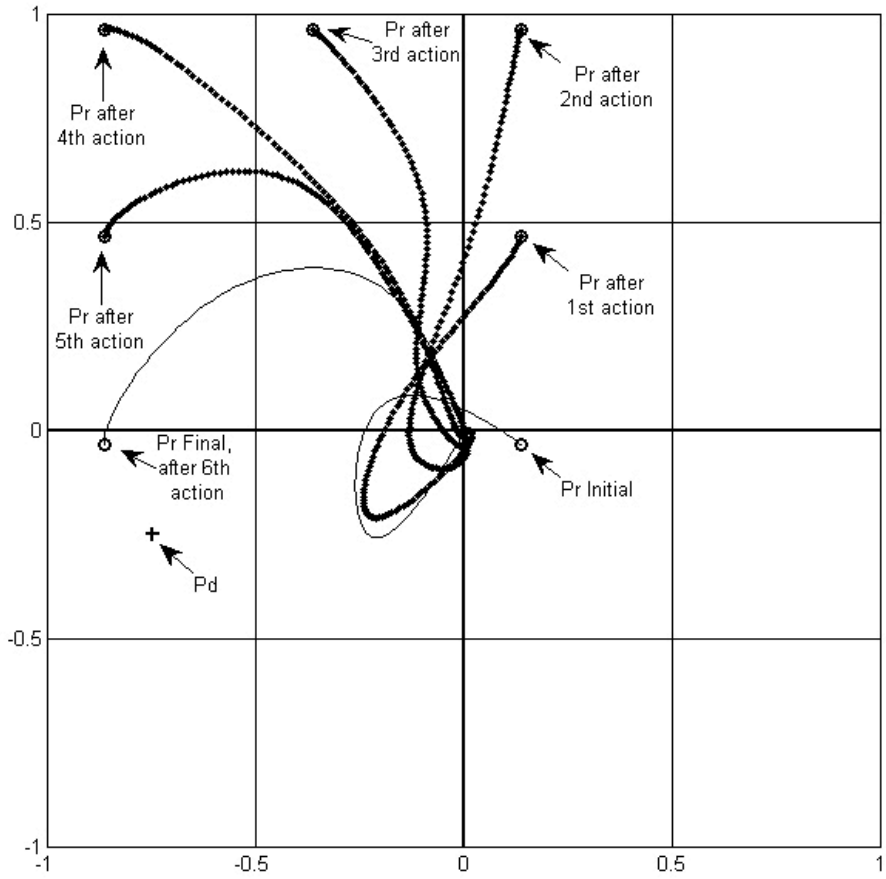


Figure 2.1: Example of an episode evolution after learning the control algorithm for a 1 robot system. Pr initial corresponds to the initial position of the tip of the hose. Pr final is the final position of the hose. The hose spline model is shown at each step.

of variable length.

Working on the GEDS model increases the computational load several orders of magnitude, therefore it has been used more as a means of validation, or to provide a realistic confirmation of the results obtained with some approach. The linear model allows more exhaustive exploration of the effect of parameter settings or structural assumptions. In one specific work (reported in Chapter 7), we use the linear model as a training arena to obtain a preliminary control algorithm which is refined on the GEDS model obtaining a substantial reduction in the learning time convergence and in the computation time invested.

Another important aspect is the trade-off between the size of the state space and the coarseness of the state representation. A very detailed representation of the hose should allow the agent to learn an optimal control algorithm, but the state space would be not be manageable for RL algorithms. A coarse representation of the state should produce a more tractable problem, but the learnt policy is likely to be sub-optimal. Some middle-ground should be found, with an acceptable compromise between state space and policy's quality.

2.3 Over-constrained systems

Reinforcement Learning (RL)[106] has been successfully 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. In this Thesis we have followed approaches derived from Q-learning [122]. Agents are the subject of learning processes trying to find the optimal action selection policy. They explore the state-action space updating the tabular representation of the state-value function according to the rewards received at the end of the learning episodes. This is a scheme of delayed rewards, which means that agents receive null rewards until some relevant terminal state is reached.

Breaking a physical constraint implies reaching states from which the learning episode cannot continue, and whose reward is not positive. This implies that, whenever one of the physical constraints is broken, the episode must be restarted, and no positive action value update is performed, in fact it can "undo" some of the updating already done. This severely reduces the probability of reaching the goal by random exploration of the state-action space. This is a critical issue, because RL algorithms rely heavily on a reasonable chance to do so. We say that a system is over-constrained when the number of such physical constraints makes no-negligible the probability of breaking some.

Rewards partition the state space S into three disjoint sets: goal states (G), transitory states (T) undesirable terminal states (U), such that $G \cup U \cup T = S$ and $(G \cap U) = (G \cap T) = (U \cap T) = \emptyset$. Formally,

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

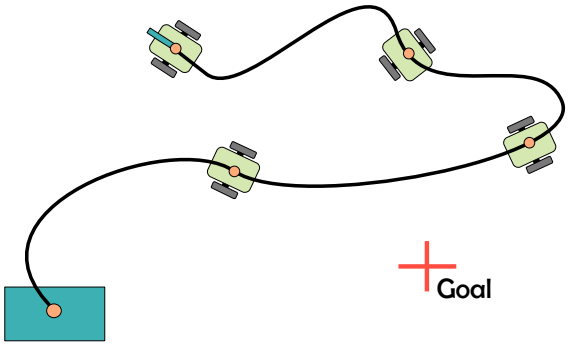
All states $s \in G \cup U$ are terminal states which force the learning episode to restart. In over-constrained systems, the size of U is not marginal respect to the size of $G \cup T$, thus the probability of randomly falling in U is non-negligible. The goal of the agent is to learn the optimal policy for every state in S. Q-Learning estimates the state-action value repeatedly visiting and updating each state-action pair. Figure 2.2 represents examples of the three different classes of states. In Figure 2.2a, a transitory state is represented, all the robots can move, no physical constraint is broken, and the tip of the hose has not reached the goal position. Figure 2.2b represents an undesirable terminal state in which one of the robots has driven over the hose. In Figure 2.2c, the tip of the hose has been correctly transported to the predefined goal position, thus reaching a goal state.

2.3.1 Some results on a single-robot system

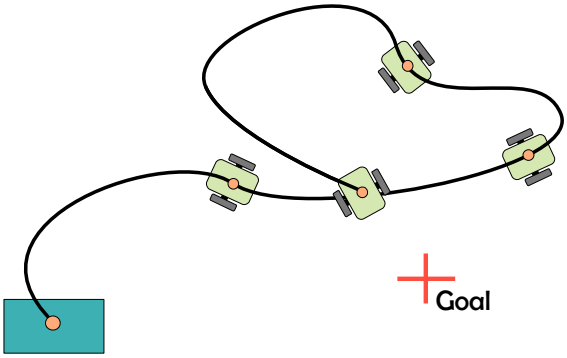
To assess the effect of the constraints in the RL performance, we did computational experiments on a single-robot instance of this hose manipulating task, using standard Q-Learning to learn the control [38] on a GEDS model of the system. Though the system is simple, it can be considered over-constrained. In order to illustrate the behavior achieved in the Q-learning computational experiments, we have chosen a difficult initial configuration in which the hose is placed between P_r and P_d . In figure 2.1 we show the successive P_r positions of the robot moving the tip-of-the-hose after each of the actions taken during the episode. The initial hose configuration corresponds to a continuous line. All the intermediate hose configurations, until the robot reached the desired cell P_d, are shown as dotted lines. It can be easily appreciated how the robot avoids colliding the hose by taking an initially suboptimal strategy (i.e. going away from the goal position). Some video animations are available in [1].

The optimal policy learnt by Q-Learning was only able to reach the goal in 73% of the randomly generated initial hose configurations after 76.000 training episodes. The 0.7% of the test episodes concluded because the maximum allowed step count was reached, and, finally, 26.3% of the test episodes failed either because the robot collided with the hose or because the whole system reached a non-feasible position.

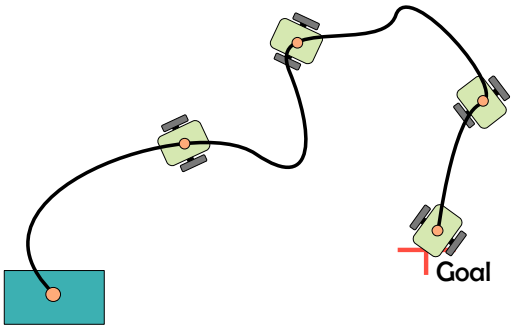
[1] http://www.ehu.es/ccwintco/index.php/DPI2006-15346-C03-03-Resultados#videos



(a) Example of a transitory state $s \in T$.



(b) Example of an undesired termination state $s \in U$.



(c) Example of a goal state $s \in G$.

Figure 2.2: Hose transportation. Examples of the three different state subspaces T, U and G.

2.4 Multi-Agent RL

Equating agents to robots, training of decentralized control of MCRS can be viewed as an instance of Multi-Agent Reinforcement Learning (MARL) systems. We consider that L-MCRS control is realized by cooperative Multi-Agent systems designed to maximize the collective utility of the system as a whole [19, 60, 88]. There is almost no literature on the application of MARL to learn the control of MCRS-like systems. The main issues of MARL from the point of view of this Thesis are scalability and non-stationarity.

Scalability refers to the ability of MARL to cope with growing size of the system and environment. In MCRS and L-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. To obtain state space and state-action space growing linearly with the number of robots the only feasible way is to produce decentralized representations and learning algorithms. When each robot has its local state-action function representation, its Q-table, and there is no global representation, the size of the complete system representation increases linearly with the number of robots, because each new one only adds a Q-table.

When all agents are learning independent and concurrently, the system becomes non-stationary from the point of view of a single agent. The local optimal policy at a given time instant can become sub-optimal in the future because of the changes of the remaining agents. To avoid non-stationarity issues, we have followed systematically an strategy for conversion of the concurrent action selection into a sequential process: we have imposed a Round Robin (RR) schedule to allow agents to select local actions, perceive local rewards (global in the case of fully cooperative systems), and update their local Q-tables. This approach can be well founded in some cases, allowing convergence proofs.

Chapter 3

Reinforcement Learning

In this chapter we give the necessary background on Reinforcement Learning (RL) algorithms covering the most conventional definitions, discussing the issues of its application to the autonomous learning of the control of Multi-Component Robotic Systems (MCRS), specifically Linked MCRS (L-MCRS), and some general discussion of the research topics in Multi-Agent RL (MARL) related to the development of the Thesis. 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, which have been explored one way or another in this Thesis. The main categories of these approaches deal with alternative system models, valuation functions, task decomposition and ways to structure the learning process.

First, Section 3.1 gives some basic definition of RL and Q-Learning. Section 3.2 introduces some alternative system models. Section 3.3 discussed the idea of functional approximation. Section 3.4 reviews approaches to automated task decomposition. Section 3.5 discusses endowing the agent with internal structure. Section 3.6 introduces the Cooperative Multi-Agent RL. Section 3.7 discusses the issues of applying RL and MARL for learning MCRS control. Section 3.8 introduces transfer learning.

3.1 Reinforcement Learning

Markov Decision Process Single-Agent RL methods model systems as Markov Decision Processes (MDPs), defined by a tuple $\langle S, A, P, R \rangle$, 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 $R : S \times A \times S \rightarrow \mathbb{R}$ is the expected reward $R(s, a, s')$ after taking action a in state s and arriving to s'. The policy applied by the agent to select the action performed at each state is modeled 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:

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\right\}, \tag{3.1}$$

where s_t and r_t are the observed state and reward at time-step t, $E_{\pi}\{.\}$ denotes the expected accumulated rewards 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:

$$V^{*}(s) = \max_{a \in A}\left\{\sum_{s'} P(s,a,s')\left[R(s,a,s') + \gamma V^{*}(s')\right]\right\}. \tag{3.2}$$

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

$$Q^{\pi}(s,a) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s, a_t = a\right\}, \tag{3.3}$$

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

$$Q^{*}(s,a) = \sum_{s'} P(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q^{*}(s',a')\right]. \tag{3.4}$$

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:

$$\pi(s,a) = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}}, \tag{3.5}$$

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. RL deals with the discovery of the optimal policy from the

Algorithm 3.1 Standard single agent Q-learning

Initialize $[Q_0(s,a); s \in S; a \in A]$ arbitrarily
Repeat (for each episode) n:

- Observe current state s_n
- Select and execute action a_n
- Observe reward r_n and new state s'
- Update the Q-value
  - $Q_n(s,a) = (1-\alpha_n)\, Q_{n-1}(s,a) + \alpha_n \left[r_n + \gamma \max_b Q_{n-1}(s',b) \right]$ if $s = s_n$; $a = a_n$
  - $Q_n(s,a) = Q_{n-1}(s,a)$ otherwise

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.

Q-Learning Q-Learning, discovered by Watkins [122], is a model-free Temporal Difference RL. It allows agents to find the optimal policy for a MDP without requiring actual knowledge about the transition function P and the reward function R, through an iterative estimation of the state-action value function that produces a sequence of estimations $\{Q_n(s,a); n = 0, 1, 2, ...\}$ converging to $Q^*(s,a)$.

The original tabular Q-Learning algorithm stores in tabular form state-action values $Q(s,a)$. Learning proceeds by a sequence of episodes as described in Algorithm 8.1. At each episode, the agent observes the actual state s_n, selects an action a_n to be executed, receives the reward r_n, observes the subsequent state s' and updates the state-action value matrix to $Q_n(s,a)$ from the previous estimation $Q_{n-1}(s,a)$. Learning speed is controlled by the gain α_n.

In [122] convergence of Q-learning in a stationary environment to an optimal policy with probability 1 is proved under certain conditions, namely that all system's states are visited infinitely often, and that the learning gain α_n converges to zero smoothly and fast enough. The Q-learning storage space requirements grow with the state and action space sizes, being of the order of $|S \times A|$. Time until convergence to the optimal policy is also growing exponentially with the

space complexity.

In the original formulation of Q-learning, the simulated/observed episodes need not follow a continuous sequence of states. In some control applications, an episode may consist of a sequence of actions leading to a successful/failing completion of a proposed task. In these realizations of Q-learning, some policy must be applied to select the sequence of executed actions. The straightforward approach (*greedy* action selection) involves selecting always the action with the highest Q-value *exploiting* available knowledge, preventing the agent *exploration* of yet unknown *action-state* pairs and, thus, hindering it to discover potentially better actions. The compromise between *exploration* and *exploitation* is typically solved using a $\epsilon - greedy$ action selection algorithm:

$$\pi^n_{\epsilon-greedy}(s,a,\epsilon) = \begin{cases} \frac{\epsilon}{|A|} & if\ a \neq \arg\max_{a' \in A} Q_n(s,a') \\ 1-\epsilon & otherwise \end{cases}. \tag{3.6}$$

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* [88] 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) [19]. 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.

3.2 Alternative MDP System Models

Several modeling enrichments developed in the context of single-agent RL can be extended 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, so they remain single-agent approaches. They can be embedded in MARL systems, as we propose in some of the chapters of this Thesis.

3.2.1 Constrained MDPs

When RL is applied to MCRS, or especially L-MCRS, broken constraints usually involve transitions to states from which the agent-environment interaction is not able to follow. These kind of termination conditions are considered *undesirable terminal states* (aka *error states* and *critical states* in the literature) in RL. They have been dealt with using null state transitions and/or generating negative rewards whenever one of those states is reached [29, 58, 21]. This is not practical in environments with a large number of undesirable terminal states. Negative rewards may discourage revisiting already observed undesirable states, however exploration policies, such as the ϵ-greedy selection, will have a non-negligible probability of revisiting them, though they do not contribute any improvement in the estimation of the Q-values.

To avoid this loss of computational/experimental resources, [46, 44, 45] propose MDPs with Constraints, where the goal is to maximize the expected accumulated reward combined with a second signal $\bar{r}$ measuring the risk of ending in an *error state* after taking action a in state s, denoted $\bar{Q}(s,a)$, calculated by an update rule analogous to the update equation in Algorithm 8.1. The state-action value is computed as the linear combination of the reward and risk Q-values:

$$Q_{\xi}^{\pi}(s,a) = \xi Q^{\pi}(s,a) - \bar{Q}^{\pi}(s,a),$$

according to the risk notion introduced by [56]. The approach in [46, 44, 45] starts exploration of the task with $\xi = 0$ until a minimum risk policy is calculated. Then ξ is progressively increased until the policy reaches a target upper bound ω of the risk of reaching an error state.

Another approach[1] assumes availability of experience tuples $< s, a, r, s' >$ from a demonstration of the task performed by an expert. 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, [55] considers undesirable states as critical and propose a graph-based algorithm to safely explore a given state-space without reaching a *critical state*. They assume that fatal states are defined by the magnitude of the rewards, which involves that states *near* critical states can be detected by the rewards. This is not always feasible and implies that the system designer chooses a correct signal related to the source of the danger. Furthermore, this algorithm is only suitable for deterministic MDPs with a small number of states.

3.2.2 Modeling action duration

The basic MDP model assumes 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 [90, 75, 17] 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 $R(s, a, s', N(a))$, 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 [107] 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 $I \subseteq T$. 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 $\mathcal{O} = \underset{s \in S}{\cup} \mathcal{O}_s$. 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 scale to large and complex problems.

3.2.3 Partially Observable Models

The focus of partially observable models is the inability 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) [64] 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[12], and only a Dynamic Programming algorithm has been proved to optimally solve them[11]. 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[99].

3.2.4 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 [20] and soft-aggregation methods to map state projections [104, 83]. Fuzzy theory has also been applied to obtain abstractions of state sets and generalize over them [9, 10, 32]. Some authors have also empirically studied different manually set state abstraction operations, such as [43] 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. [74] 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 coordination/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 [49]. 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 [61]. 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. [23] 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 sub-tasks, yielding higher scalability.

3.3 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 VFAs: Local Linear Regression [109], weighted Radial Basis Functions [70], Cerebellar Model Arithmetic Computers [15, 108], Artificial Neural Networks [26, 124], instance-based approximators[3], and Least Squares Policy Iteration[53]. It has been discussed whether VFAs might be appropriate in the general case (in favor [108], against [16]), because of the assumptions on the topology of the functions [105]. They remain to be applied to MARL systems. VFA 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.

3.4 Automatic Task Decomposition

After manually designed task decomposition was successfully applied to increasingly complex environments [91], the automatic decomposition of tasks became a hot subject and it has thereafter focused great scientific interest [24], because of the inherent scalability of automated approaches.

A medium level of automation is introduced in [102], which proposes defining some basic MAXQ hierarchy to introduce domain knowledge in the system and using options to learn sub-tasks 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* [57], 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 [22, 77, 84, 103] 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 sub-tasks in a transition-graph is commonly considered as a clustering problem, implicitly assuming state clusters are considered sub-tasks to be solved towards reaching sub-goals, which have been identified as states with a high reward gradient [31] or states that are often visited on successful trajectories [81], 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 sub-task. The *Q-Cut* algorithm [84] 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)[8] 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* [103] was presented as a more scalable partitioning 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, [23] 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. [77] 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) [62, 63] 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 [81], identifying sub-goals as small sets of states often visited in successful trajectories.

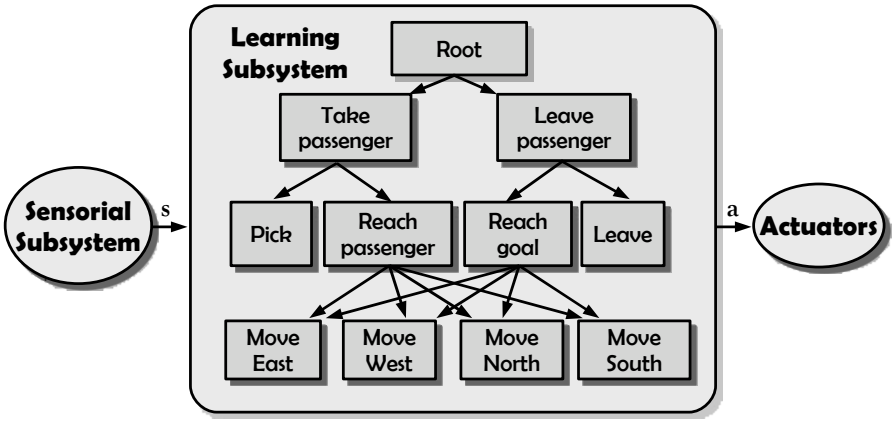


Figure 3.1: Example of Hierarchical RL on the taxi domain.

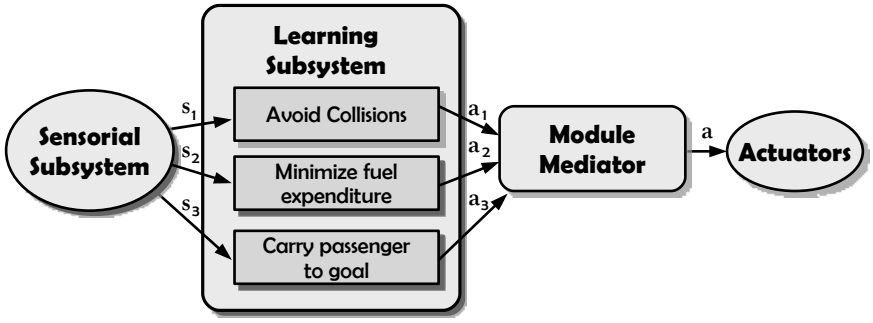


Figure 3.2: Examples Modular RL for the classical taxi driver problem.

3.5 Structured Single Agent RL

Learning processes can be simplified by decomposing tasks into more manageable sub-tasks, 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 sub-tasks are appropriately defined. There are two main Structured RL approaches: Modular RL (MRL) and Hierarchical RL (HRL). The former considers executing concurrent sub-tasks, while the latter defines a hierarchical structure of tasks. To illustrate the approaches we will consider this variant of the classical *taxi driver problem* (Appendix B).

3.5.1 Modular RL

Modular RL (MRL) [13] decomposes complex problems into a collection of m simultaneously running simpler learning processes with potentially disjoint state subspaces. Action decision is based on the output of these modules, which may receive specific reward signals. In the general case, learning modules share a common set of actions from which to choose but have their own reward signal and state subspace[1]. Formally, the agent's state space S is composed $S = \times_{i=1}^{m} S_i$, where $S_{i,\, i=1,\ldots,m}$ is the local state subspace of the i^{th} module. This approach reduces the size of the state-action space needed to learn each of the sub-tasks, but requires some additional mechanism to produce a unique agent action. Figure 3.2 shows a modular structure where each module can propose a single action, and a *Module Mediator* (*Module Arbiter*) is responsible for selecting a unique action.

Other approaches let each module assign a numerical value to each of the eligible actions. These numerical values can be locally estimated $Q_i(s_i, a)$ values. Agent action selection policies, such as the Greatest Mass (GM) strategy [85, 123], can be defined on these local state-action tables:

$$\pi(s_i) = \arg \max_{a \in A} \left\{ \sum_{i=1}^{m} Q_i(s_i, a) \right\}, \tag{3.7}$$

where $s_i \in S_i$. In general, modular action selection policies $\pi : \{A, \mathbb{R}\}^m \rightarrow A$, allowing the i^{th} module to propose a single action $a_i \in A$ with an associated weight $w_i \in \mathbb{R}$ can be defined [54]. 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. Alternatively, other approaches use *gating signals* to decide which module is designed responsible in each state [111, 98], or share the reward among modules [125].

3.5.2 Hierarchical RL

Whereas MRL deals with concurrent tasks, HRL decomposes complex tasks into sequentially executed simpler sub-tasks which are executed from the upper sub-task in a recursive manner. Figure 3.1 shows one such a hierarchical decomposition for the taxi problem. Subsystems performing these sub-tasks are to be separately trained in order to solve the global task.

MAXQ algorithm. One of the most extensively used HRL algorithms is *MAXQ* [29, 30], which is based on HAMQ algorithm [89, 90] and decomposes a MDP into a set of sub-tasks 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 sub-task M_i is terminated, A_i is a set of allowed actions during execution of

[1] Different modules could share state variables. To simplify notation, we will assume disjoint subsets without loss of generality.

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, namely, the expected reward received from executing sub-task a, denoted $V(a,s)$, and the expected received reward from the end of sub-task a until the completion of parent task M_i, which is also know as the *completion function*:

$$C^{\pi}(i,s,a) = \sum_{s',N} P_i^{\pi}(s', N \mid s, a)\, \gamma^N Q^{\pi}(i, s', \pi(s')), \tag{3.8}$$

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

$$Q^{\pi}(i,s,a) = V^{\pi}(a,s) + C^{\pi}(i,s,a), \tag{3.9}$$

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

$$V^{\pi}(i,s) = \begin{cases} Q^{\pi}(i,s,\pi_i(s)) & if\ Composite\ (M_i) \\ \sum_{s'} P(s' \mid s,i)\, R(s' \mid s,i) & if\ Primitive\ (M_i) \end{cases}. \tag{3.10}$$

This decomposition allows a compact representation and only requires to store the Q function for composite sub-tasks 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 sub-tasks. Shared subtasks is another added advantage: for example, subtasks *Reach-Passenger* and *Reach-Goal* in the Figure 3.2 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, [76] 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 known 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 [47]. 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.

3.5.3 Hybrid Structures

Trying to have both the concurrent computation of modular structures and the task decomposition of hierarchical structures, [95] proposes concurrent options using disjoint action spaces which don't interfere with each other. Another hybrid approach is to define hierarchies of module groups [110], 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.

3.6 Cooperative Multi-Agent Reinforcement Learning

When the system is composed of several interacting autonomous agents, the Multi-Agent Reinforcement Learning (MARL) problem consists on the search for the optimal policies for each agent [19]. The extension of the MDP for the Multi-Agent case is a Stochastic Game (SG) defined as a tuple[2] $< S, A_1, \ldots, A_N, \mathbf{P}, \mathbf{R}_1, \ldots, \mathbf{R}_N >$, 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 the agents execute the joint action $\mathbf{a} \in \mathbf{A}$ in state s, and $\mathbf{R} : S \times \mathbf{A} \times S \rightarrow \mathbb{R}$ are the expected rewards received by the 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 form a global *joint policy* $\boldsymbol{\pi}(s, \mathbf{a}) = \{\pi_1(s, a_1), \ldots, \pi_N(s, a_N)\}$. The local Q-function of each agent depends on the global action and is conditioned by the global policy $\mathbf{Q}_i^{\boldsymbol{\pi}} : S \times \mathbf{A} \rightarrow \mathbb{R}$. The goal of the MARL is to learn the optimal policies maximizing the accumulated rewards received by all agents.

[2] 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.

If the local rewards are identical for all agents $\mathbf{R}_1(s, \mathbf{a}, s') = \mathbf{R}_2(s, \mathbf{a}, s') = \ldots = \mathbf{R}_N(s, \mathbf{a}, s')$, the system is fully *cooperative*. When the rewards have opposite signs for different agents, then we have competition. We are concerned in this Thesis mainly with cooperative systems. 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 [112], 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 [19] for a more in-depth review.

3.6.1 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.

Precursor of more advanced algorithms, *Distributed Rewards* and *Distributed Value Functions* were studied in [100] 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. These methods scale linearly to the number of agents, but offer no guarantees of optimality.

Distributed Q-Learning In *Distributed Q-Learning (D-QL)* [72] each agent makes optimistic assumptions about the behavior of the remaining agents, assuming all of them are choosing the optimal action determined by the actual state-action values given by the Q-table. Joint state-action Q-table is decomposed into smaller local Q-tables storing only the best values of the joint state-action Q-table $\mathbf{a}$ constraining the local action $a_i = a$

$$Q^i(s, a) = \max_{\substack{\mathbf{a}=[a_1 a_2 \ldots a_N] \\ a_i = a}} \mathbf{Q}(s, \mathbf{a}),$$

where $\mathbf{Q}(s, \mathbf{a})$ represents the virtual state-action value of joint action $\mathbf{a}$ of a virtual centralized learner, and $Q^i(s, a)$ is the state-action value locally stored by the i^{th} agent. This method needs only be aware of the local action, and updates the local Q-values only when the updated state-action value is higher than the previous one:

$$Q^i_t(s, a) = \begin{cases} Q^i_{t-1}(s, a) & if\ s \neq s_{t-1}\ or\ a \neq a_{t-1} \\ \max\left\{Q^i_{t-1}(s, a), r_t + \gamma \max_{a'} Q^i_{t-1}(s', a')\right\} & otherwise. \end{cases}$$

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

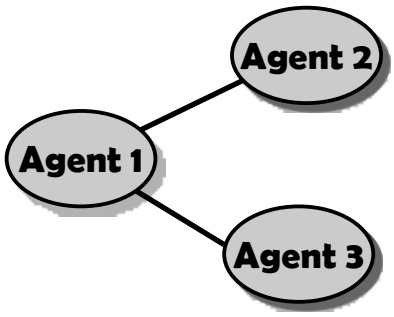


Figure 3.3: Coordination Graph example

any explicit communication between agents. In this paper, the original D-QL algorithm is the benchmark for comparison with our proposal.

Several other distributed approaches can be found in the literature. For example, some authors propose the use of an actual centralized table updated by all agents [28, 14]. Some others have presented very good results applying a heuristically modified version of D-QL, known as Hysteretic QL [80]. Distributed RL has also been proposed to approach multi-objective optimization problems using negotiation protocols between agents [78].

3.6.2 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 [94]. Some authors have proposed several heuristic algorithms [25, 66] to estimate the most likely response of the rest of agents using models. 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 [83, 120] 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.

3.6.3 Coordinated MARL

The coordinated MARL performs 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 [35]. *Distributed Rewards* and *Distributed Value Functions* were studied in [100] as a way to stimulate and control cooperation between neighbors. Instead of updating the state-action values using only the local reward or value functions, agents also use 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 RL* (Coordinated-RL) [53] approximates the global joint value function as a linear combination of local value functions [52]. The complexity of agreeing on a globally optimal joint action can be reduced assuming that agents need not 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 3.3. 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 Figure 3.3, the coordination task is to find the joint actions that maximize the joint reward given by the addition of the individual rewards:

$$(a_1^*, a_2^*, a_3^*) = \arg \max_{(a_1,a_2,a_3)} \{Q_1(s_1, a_1, a_2, a_3) + Q_2(s_2, a_1, a_2) + Q_3(s_3, a_1, a_3)\}. \tag{3.11}$$

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 [118] 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 a 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 known 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 [69]: 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* [68] (or SparseQ) is a distributed version of the traditional Q-Learning for which two global value function decomposition methods have been proposed [69]: 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.

3.7 RL and MARL for MCRS control: a discussion

The main advantage of adopting the MARL framework for the development of MCRS control algorithms is that it provides 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. Some, such as coordination-related issues, are specific to MARL algorithms, others which are inherited from the basic single-agent RL methods, 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 order of the table size is $O(\mid S \times A \mid)$, getting even worse in most MARL algorithms, where it grows exponentially as the number of agents increases: $\mid S \times A^n \mid$. This problem is the manifestation of the *curse of dimensionality* in RL, and it is the most serious limit to scale up single agent RL Q-Learning to MARL systems. Besides, communication resources needed for RL also scale up combinatorially with the number of robots/agents. Single-agent

RL requires knowledge about chosen actions to be broadcasted 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. Hierarchical solutions [47] can be considered to face this problem.

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 with VFA 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 environmental 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: In conventional 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 involve 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. 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: 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 the convergence of *Stochastic Games* to the *Nash Equilibrium* [19]. If each agent follows an op-

timal 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.

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 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 consumes time and adds complexity to the synchronization issue. Coordination algorithms [53, 69] may introduce complex communication protocols to agree on a joint action to be taken.

No relevant literature can be found regarding this issue in RL algorithms. This is a big issue in coordinated MARL algorithms, even using anytime algorithms. 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 $t = t_0$, predict the state 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 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.

The use of POMDP 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 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 [19].

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.

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.

3.8 Transfer Learning

Based on the concept of incremental learning, Transfer Learning (TL) [114] aims at speeding up the learning of a *target task* using available knowledge from a *source task*, somehow related to the former. In a RL context, TL means taking advantage of knowledge learnt from a *source MDP* $< S', A', P', R' >$ (i.e. *low-level* knowledge such as Q-values or *high-level* knowledge such as options [107]) to improve the learning of a *target MDP* $< S, A, P, R >$. This idea is depicted in Figure 3.4. The source task is always a simplified model that can be used to obtain optimal policies easier than with the target task. The basic TL process can be summarized in three sequential steps: first, the agent learns the source task; second, the acquired knowledge is transferred to the target task and, finally, the agent learns the target task.

Several different approaches can be found in the literature [114], each of them can be distinguished by the following features: differences allowed between source and target MDPs, how source tasks are selected, how to map different action and/or state spaces, the way knowledge is transferred, the allowed learners, and the metrics used to measure improvement.

One of the earliest works applying TL for RL [101] studied the change of the transition function in time, namely, an agent first learns to balance a light

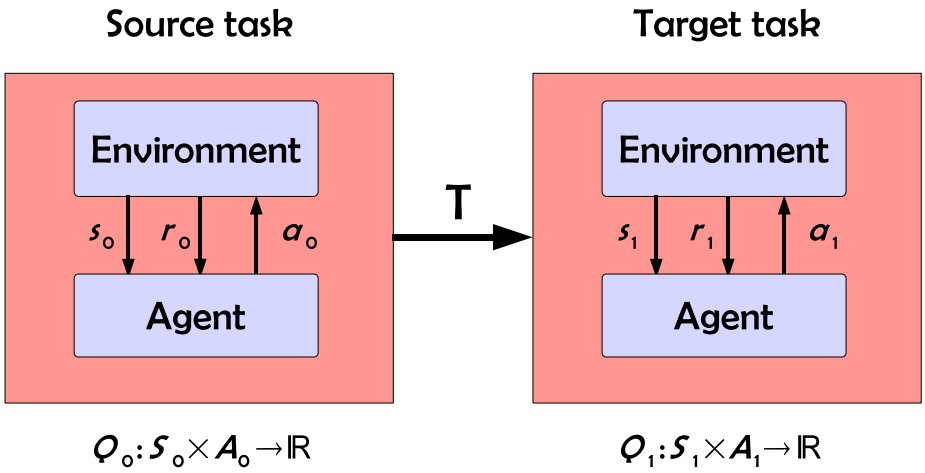


Figure 3.4: RL Transfer Learning scheme

and small pole in the classical cart-pole problem and then the learned policy is used for the control of a longer and heavier pole. [7] studied the use of learning from a simulated environment and then transferring learnt knowledge to the real environment. These two works proposed Q-values to transfer experience, [73] transferred experience tuples $< s, a, s', r >$ between tasks. While both Q-values and experience tuples can be considered low-level knowledge, some other authors have considered transferring higher-level knowledge such as Proto-value functions [37], which represent the structure of the task or subroutines [4]. In all these cases, the authors considered a fixed set of state variables for both source and target tasks. Transfer between different state spaces have also been explored [115, 113, 51]. Relational RL approaches [93, 27] learn and build first-order rules, which can be easily exported to new environments. Most of the literature validates TL methods empirically and the methods are problem-specific, therefore it is not possible to provide a systematic comparison between methods.

Chapter 4

Thresholded MSAV for Multi-agent-POMDP

In this chapter we introduce the Thresholded Modular State-Action Vetoes (Thresholded-MSAV) to deal with over-constrained systems, in a very specific instance of the L-MCRS control problem given by the simplified hose model of Appendix B. The Thresholded-MSAV presentation is intuitive without formal proofs of convergence. Modular agents learn several sub-tasks concurrently, and our approach uses one module to learn the main task while the others learn the constraints, which trigger negative rewards when they are broken. Besides, it is embedded in a multi-agent architecture, which is also appropriate for L-MCRS control problems. We propose Partially Observable Markov Decision Problems (POMDP) as an scalable representation of the task to be learnt. Moreover, we propose a Round Robin (RR) realization of the action selection process to avoid non-stationary situations. Agents use Thresholded MSAV module based Q-learning to make a more safer and more efficient exploration of the state-space.

The chapter is structured as follows. Section 4.1 gives an introduction. Section 4.2 defines the Round-Robin POMDP framework. Section 4.3 defines the modular agent. Section 4.4 proposes an algorithm to learn the Q-values in the modular agent. Section 4.5 describes the Thresholded-MSAV approach. Section 4.6 shows the results of our computational simulations for two different degrees of cooperation between the agents. Finally, we provide the conclusions in Section 4.7.

4.1 Introduction

Learning in a over-constrained environment can be decomposed as two concurrent sub-tasks: learning the main task and learning to avoid breaking the constraints. This decomposition naturally leads to Modular RL, where one RL module learns the main task, while the rest learn to avoid breaking the constraints. The system is able to learn early avoidance of dangerous state-actions using *Modular State-Action Vetoes* (MSAV). Modular decomposition of the problem offered an additional advantage: each sub-task can be learnt using only task-relevant state variables, reducing the dimensionality of the learning problem at each module. Decentralized control algorithms require multi-agent approaches with several agents, each of them controlling one of the robots. This worsens the dimensionality issue. To alleviate it, we use two different techniques: POMDPs and RR scheduled execution of actions.

The topology of the robots in a L-MCRS has certain features that can be exploited to reduce the information requirements in a multi-agent environment. For example, in the hose transportation problem, robots are attached to the hose following a sequence along it. Our working hypothesis here is that each agent needs not know the position of all the robots to effectively control its assigned robot towards a predetermined goal position. We have investigated whether local information about the neighbors can suffice to solve the problem.

Using RR cycles of executions, agents can take actions and perform (concurrent) independent learn processes. This method allows agents to learn without having to explicitly coordinate with other agents. During an agent's turn, the environment becomes stationary from its viewpoint, thus avoiding non-stationarity that plagues MARL algorithms.

4.2 Round-Robin POMDP

Definition 1. A *Round-Robin Partially Observable Markov Decision Process* (RR-POMDP) is a tuple $< S, \Omega_1, \ldots, \Omega_N, A_1 \ldots A_N, P, R_1, \ldots, R_N, \delta >$, where

- N is the number of agents.
- S is the set of states spanned by state variables X.
- $\Omega_i : S \rightarrow S_i$ is the state subspace perceived by the i^{th} agent. This is a projection of the state space to a subset of the state variables $X_i \in X$. For notation simplicity, we assume disjoint subsets of state variables and, thus $S = \times_{i=1}^{N} \Omega(S)$.
- A_i is the set of i^{th} agent's local actions. The complete set of actions is therefore $A = \underset{i=1 \ldots N}{\cup} A_i$.
- $P : S \times A \times S \rightarrow [0, 1]$, $i = 1, \ldots, N$ is the state transition function $P(s, a, s')$ that defines the probability of observing s' after agent i executes action a selected from its local action space A_i.

- $R_i : S \times A_i \times S \to \mathbb{R}$ is the shared scalar reward signal received after execution by any agent of an action from A_i.
- $\delta : \mathbb{N} \to \mathbb{N} \in [1, N]$ is the function defining the cycle of action execution. $\delta(t)$ gives the index of the agent allowed to take an action in time t (times are relative to the start of the episode). This function is cyclic, therefore $\forall t \in \mathbb{N};\ \delta(t) = \delta(t+N)$.

At time t, $i = \delta(t)$ determines the agent allowed to execute an action following its own local policy and we assume all local policies share the same learning parameters.

We distinguish two kinds of systems: the fully-cooperative case when $R_1(s) = \ldots = R_N(s)$, and the uncooperative case when each agent has its own goal and, thus, its own different reward signal $R_i(s)$. The cooperative system will be used to model multi-agent systems with a common goal, while the uncooperative will be suitable for problems where multiple loosely-dependent objectives are pursued by the agents.

4.3 Modular RR-POMDP

Each agent of the RR-POMDP is decomposed into a set of RL-modules. If we model the i^{th} agent as a Monolithic MDP, it would have sate space S_i and would receive a reward signal R_i. In the modular decomposition the i^{th} agent uses a set of m modules. Each j^{th} module has state space $S_{i,j}$ and receives a reward signal $R_{i,j}$. We formally define this decomposition in the following paragraphs.

Definition 2. A *modular RR-POMDP* is defined as a set of N agents, each of them implemented as m concurrent independent modules with disjoint state subspaces, different rewards, and a common action space. We denote it as $\{< S_{i,j}, A_i, P, R_{i,,j} >^{N,m}\}$.

In the definition of a RR-POMDP, we assumed disjoint state variables for each agent. Likewise, we assume disjoint state variables for each module.

Definition 3. A Modular RR-POMDP $\{< S_{i,j}, A_i, P, R_{i,j} >^{N,m}\}$ is a decomposition of a monolithic RR-POMDP $< S, \Omega_1, \ldots, \Omega_N, A_1 \ldots A_N, P, R_1, \ldots, R_N \delta >$ if $\forall i = 1 \ldots N;\ \Omega_i(S) = \times_{j=1}^{m} S_{i,j}$, and $\forall s \in S;\ \forall i = 1 \ldots N; R_i(s) = \sum_{j=1}^{m} R_{i,j}(s)$.

Our approach considers two different types of modules for each agent $i = 1 \ldots N$: a *Goal Module* $< S_i^G, A, P, R_i^G >$, which learns the main task, and $m-1$ different *Veto-Modules* $< S_{i,j}^U, A, P, R_{i,j}^U >$ corresponding to each class of undesired termination states. Each module observes a different disjoint subset of the state variables and receives an specific reward signal. The reward signals $R_i^G(s)$ and $R_{i,j}^U(s), i = 1, \ldots, N,\ j = 1 \ldots m-1$ are fed to the Goal Module and Veto Modules, respectively, as represented in Figure 4.1.

- $R_i^G(s) \geq 0$ is strictly positive only if $s \in G$.

- $R^U_{i,j}(s) \leq 0$ are strictly negative signals only when a certain class of undesired state has been reached i.e. collision, broken physical constraint, etc.

Assuming that only one undesired type of state can be reached each time-step,

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

leading to a decomposition of a given RR-POMDP that fulfills Definition 3.

4.4 Q-Learning in a Modular RR-POMDP

Each i^{th} agent's j^{th} module stores its local state-action values as a matrix $Q^i_j(s,a)$, $s \in S_{i,j}$, $a \in A_i$. An immediate advantage of the Modular RR-POMDP approach is the reduction of storage requirements. Q-learning in a monolithic RR-POMDP would require up to $\mid S_i \times A_i \mid$ entries to represent the Q-values, whereas a modular learner will only require $\sum_{j=1}^{m} \mid S_{i,j} \times A_i \mid$ entries. We propose to update these Q-values using Q-Learning's update rule:

$$Q^i_j(s,a) \leftarrow Q^i_j(s,a) + \alpha\left(r + \gamma * \max_{a'} Q^i_j(s',a') - Q^i_j(s,a)\right), \tag{4.1}$$

where $Q^i_j(s,a)$ can be expressed as

$$Q^i_j(s,a) = E_\pi\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}, \tag{4.2}$$

that will eventually converge to the optimal Q-values

$$Q^i_j(s,a)^* = \sum_{s' \in S_{i,j}} P(s,a,s')\left[R_{i,j}(s,a,s') + \gamma \max_{a'} Q^i_j(s',a')^*\right] \tag{4.3}$$

Equation 4.1 is analogous to the standard Q-learning update rule 8.1. The complete algorithm implemented in each module is shown in Algorithm 4.1. Note that, for the algorithm to work properly, all the rewards received by an agent in a cycle are summed before the Q-matrix is updated. This allows those rewards obtained by other agents' actions to be spread among agents, no matter the reward signal they receive is the same or not.

It is worth noting that, while Q-learning modules will converge to the optimal modular Q-values $Q^i_j(s_i,a)^*$, there is no guarantee that a modular agent's greedy policy will be optimal with respect to the RR-POMDP $< S, \Omega_1, \ldots, \Omega_N, A_1 \ldots A_N, P, R_1, \ldots, R_N \delta >$.

Algorithm 4.1 Learning algorithm used by each module j in a Modular RR-POMDP.

Initialize $\left[Q^i_{j,0}(s,a) = 0; s \in S_{i,j}; a \in A_i; i = 1 \dots N\right]$ 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, allowing a cycle of actions
- Observe state s' and following rewards $r_t, \dots, r_{t+N-1}$
- Update each module state-action value
  - $Q^i_{j,n}(s,a) = (1-\alpha_n)\, Q^i_{j,n-1}(s,a) + \alpha_n \left[\sum_{k=0}^{N-1} r_{t+k} + \gamma \max_{a'} Q^i_{j,n}(s',a')\right]$ if $s_t = s$, $a_t = a$, $i = \delta(t)$
  - $Q^i_{j,n}(s,a) = Q^i_{j,n-1}(s,a)$ otherwise

until s' is terminal

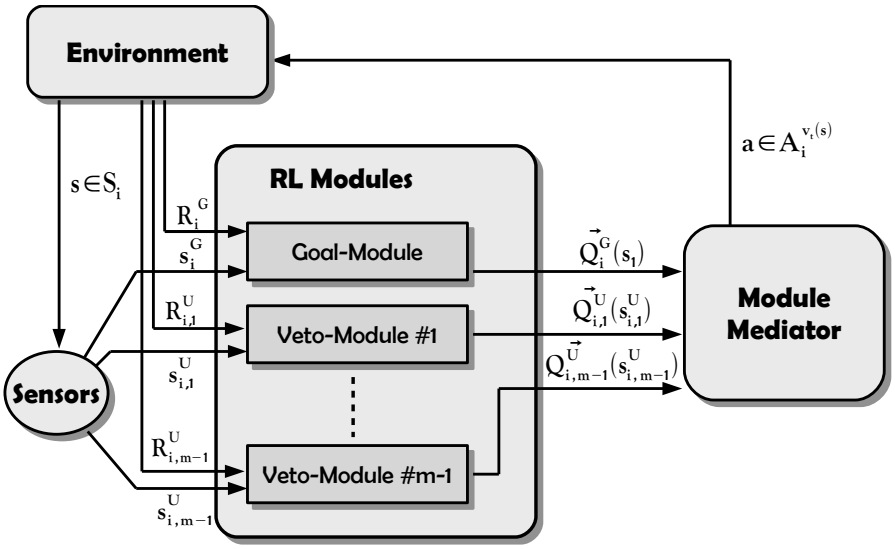


Figure 4.1: Modular decomposition of Thresholded MSAV for Q-Learning

4.5 Thresholded Modular State-Action Vetoes

The learning scheme used is depicted in Figure 4.1. Each RL module outputs its Q-values, and the Module Mediator selects an action taking into account these outputs. The Module Mediator we introduce in this chapter only considers those actions whose Q-values are above a given threshold v_t, vetoing the rest. This action selection policy is what we refer to as *Thresholded Modular State-Action Vetoes* (Thresholded-MSAV). In this section, we purposefully omit the agent index.

Definition 4. We define the *Thresholded-safe action repertoire* $A^{v_t}(s)$ as the set of actions whose optimal Q-values are above the threshold v_t:

$$A^{v_t}(s) = \{a \mid a \in A,\ \forall i = 1 \dots m;\ Q_i^*(s_i, a) \geq v_t\},$$

where $s = \{s_1, s_2 \dots s_m\},\ s_i \in S_i$.

To minimize the risk of breaking a constraint, we propose to use a simple veto system, allowing the Module Mediator to choose only from the local estimation of $A^{v_t}(s)$. An example of these Thresholded-MSAV policies can be derived from standard $\epsilon - greedy$ action selection algorithm using the Greatest Mass strategy:

$$\pi_{\epsilon}^{v_t}(s,a) = \begin{cases} 0 & a \notin A^{v_t}(s), \\ \frac{\epsilon}{|A^{v_t}|-1} & (a \in A^{v_t}(s)) \wedge a \neq \arg\max_{a' \in A^{v_t}} \left\{ \sum_{i=1}^{m} Q_i(s_i, a') \right\}, \\ 1-\epsilon & (a \in A^{v_t}(s)) \wedge a = \arg\max_{a' \in A^{v_t}} \left\{ \sum_{i=1}^{m} Q_i(s_i, a') \right\}. \end{cases} \quad (4.4)$$

This means that if, under state s, a module i has a value $Q_i(s_i, a)$ below the veto threshold v_t, action a is forbidden for that state. Traditional $\epsilon - greedy$ is then used to select an action among the available ones in $A^{v_t}(s)$, assuring this way that taking a random action will not result in an abrupt failure once the modules have a minimum amount of experience, thus allowing a more efficient exploration of the state-action space. Instead of manually tweaking the ϵ parameter, we obtain a simple approach to keep goal-focused exploration while avoiding constraint-related termination conditions.

4.6 Experiments

Computational experiments were conducted on the hose transportation task using a simplified hose model (see Appendix B) with two different degrees of cooperative agents: fully-cooperative and uncooperative. In the first case, agents share the reward signal, so local value maximization implies cooperation. In the second case, agents act selfishly trying to maximize their own local reward, which is different for each agent. Different computational experiments were performed each for one class of agent cooperation categories. Some parameters are common to both experiments. They are run with a maximum hose length L_{hose} of 16 cell units, so that each hose segment is limited to a maximum length $L_s = \frac{L_{hose}}{N}$ (expressed in cell units). The step limit count for episodes is set to 400. Episodes that don't reach the goal within this limit are forced to stop. The vetoing threshold v_t is set to -10.

4.6.1 Fully-cooperative agents

We measure the learning performance of several module combinations (including both homogeneous and heterogeneous agents). We report here the best results, which were achieved using four different modules.

***Goal* module** The state variables observed by the goal module of the i^{th} agent are d_i^g and θ_i^g. Whenever the tip of the hose reached the goal, all instances of these module in all the agents receive the same positive reward. The Q-matrices of the goal modules have a total amount of $N \cdot \mid A_i \mid \cdot \mid d_i^g \mid \cdot \mid \theta_i^n \mid$ entries.

***Veto-Distances* module** The state of the i^{th} agent veto-distance module is a combination of r_i^n and r_i^p. This module receives a negative reward every

time any of the two hose segments exceeds the maximum allowed length. The Q-matrices corresponding to these modules have $N \cdot \mid A_i \mid \cdot \mid r_i^d \mid \cdot \mid r_i^n \mid$ entries.

***Veto-Collisions* module** This module learns to avoid collisions between robots and hose segments. Its state is the combination of the four boolean flags $o_i^{up}, o_i^{down}, o_i^{left}$ and o_i^{right}, each indicating whether there is an obstacle (robot or hose segment) within one time-step reach in a direction corresponding to one of the allowed actions. Whenever a collision occurs a negative reward is received. The Q-matrices of these modules have $N \cdot \mid A_i \mid \cdot \mid o_i^{up} \mid^N$ entries.

***Veto-In-Grid* module** The robots were desired to stay within a predefined boundaries of the working space. To that end, this module's state is the absolute position of the robot controlled by the i^{th} agent, p_i. A negative reward is received if the position is outside the allowed bounds. The Q-matrices have $N \cdot \mid A_i \mid \cdot \mid p_i \mid$ entries.

Experiment A has a two-fold goal: (1) measure the scalability of the approach to a growing number of robots, and (2) quantify the effect of the MSAV in such an over-constrained environments. Five different configurations are tested with and without the MSAV ($N = 2, 3, 4, 5, 6$). Each episode is started with a randomly selected position for each robot and goal. Those random configurations that don't fulfill the system constraints are rejected. The ϵ parameter is set to a fixed value: 0.3. A total amount of $100,000$ episodes are simulated.

Experiment B goal is to evaluate the ability of the system to converge to a greedy policy. With this purpose in mind, we select a random configuration before starting simulation and the system training is repeated from this initial configuration. Initially, $\epsilon = 1$ and it is decreased 0.001 each time an episode ends successfully. Overall, $10,000$ episodes were conducted.

4.6.1.1 Storage requirements

A comparison of the storage requirements needed when the task is modeled as a MDP and a Modular RR-POMDP shows the advantages of the latter. Table 4.1 shows the cardinality of the domain of each state variable used in this experiment. Note that the cardinality for r_i^n and r_i^p is dependent on the number of robots, because in this experiment the allowed hose segment maximum length depends on the actual number of robots. Using these values, the total Q-matrix storage requirements are shown in Table 4.2 for both learning frameworks. The entries required to solve the task as a single-agent MDP were calculated as the minimum amount of entries to store all relevant information (the complete formation of robots can be represented using $N \cdot \mid A \mid^N \cdot \mid r_i^d \mid^N$ states). The huge differences are due to two different facts: the modular decomposition and the coarse state representation used in the *Goal* module (d_i^g and θ_i^g). While the storage requirements for the typical monolithic approach grows rather fast,

State variable	Domain Cardinality				
d_i^g	24				
θ_i^g	8				
o_i^j	2				
P_i	441				
	$N=2$	$N=3$	$N=4$	$N=5$	$N=6$
r_i^n,r_i^p	197	89	49	37	21

Table 4.1: Domain cardinality for each state variable for a maximum hose segment length of $\frac{L_{hose}}{N}$ cell units.

Learning framework	Entries				
	$N=2$	$N=3$	$N=4$	$N=5$	$N=6$
Single-agent MDP	$970,225$	$8.8 \cdot 10^6$	$3.6 \cdot 10^9$	$2 \cdot 10^{11}$	$1.3 \cdot 10^{12}$
Modular RR-POMDP	$393,220$	$127,050$	$59,720$	$49,750$	$35,586$

Table 4.2: Total amount of entries required to store the Q-table for this task using different learning frameworks.

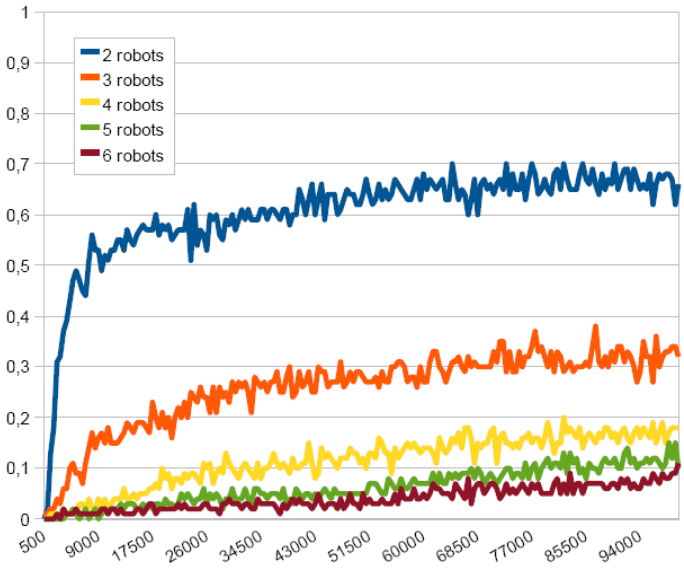
the storage requirements for the modular approach decrease as the number of robots increases.

4.6.1.2 Experiment A

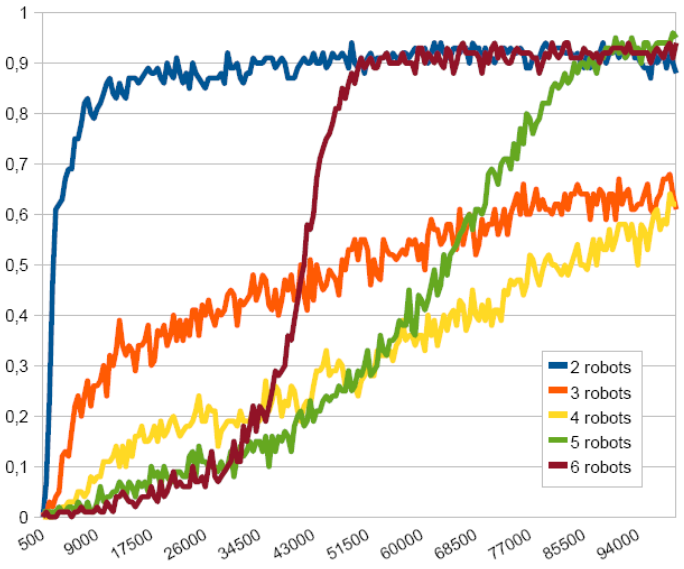
We report results on the improvement introduced by the use of the Thresholded MSAV policy versus the conventional $\epsilon - greedy$ action selection in the concurrent training of the agents. For each experiment the success rate for different number of robots is plotted[1]. Videos were generated for the execution of each of the policies for visually validation of the simulations[2]. Figures 4.2a and 4.2b show the evolution of the episode success ratio as the training evolves without and with the Thresholded MSAV, respectively, for an increasing number of robots. In both cases, increasing the number of robots dramatically reduces the capacity to reach the goal. However results are much better for the veto system (figure 4.2b) than for the baseline monolithic system (figure 4.2a).

[1]Episode success ratio is plotted in the interval $[0, 1]$, the abscissa corresponds to the episode count.

[2]Some videos can be downloaded from `http://www.ehu.es/ccwintco/index.php/Borja-videos`



(a) Episode success rates for different number of robots without Thresholded MSAV.



(b) Episode success rates for different number of robots using Thresholded MSAV.

Figure 4.2: Cooperative learning of the simplified hose transportation task: experiment A.

Upon visual inspection of the videos of the unsuccessful episodes, it became clear that failure was always due to the lack of coordination between agents. Figure 4.3 represents an instance of this kind of uncoordinated behaviors: agents controlling P_0 and P_3 try to reach the goal position, being unaware of the global system configuration, leading the system into deadlock state. In some cases, some random sequence of actions (random actions are selected with probability ϵ) leads to a configuration from which the system can successfully continue toward the goal, but there is no guarantee of reaching such configurations. The use of a subset of the state variables based on topological proximity results in such uncoordinated behaviors even in cooperative systems. The use of a shared set of state variables is expected to introduce better coordination in the case of shared rewards at the cost of more memory to represent the Q-values.

Another interesting observation is that, using Thresholded MSAV, the percentage of successful episodes is reduced as the number of robots/agents increases, but it somehow increases again for $N = 5, 6$. Our educated guess is that, since the hose segment size is reduced as the number of robots grows, the complexity of the learning problem for the *Constr-Distances* module is reduced with a bigger number of robots (Table 4.1). This complexity reduction becomes more important than the overall increased complexity of the system for $N > 5$. This result confirms the importance of the modular vetoes in such over-constrained environments.

4.6.1.3 Experiment B

In this experiment, the agents were trained repeatedly starting from the same initial configuration. This repetition reduces the complexity of the learning task and thus, experiments report better results than in the previous experiment. Figure 4.4a represents the average success rate obtained during the learning process and Figure 4.4b shows the total amount of steps needed by successful episodes to reach the goal.

Figure 4.4a shows that, as in the previous experiment, the number of episodes needed to effectively veto potentially dangerous actions decreases clearly as the number of robots is increased from 2 to 4, but when the number is further increased, the smaller size of hose segments makes learning the constraints an easier task.

Figure 4.4b shows that the system converges to an optimal policy as the ϵ parameter is reduced. This convergence is clearly faster, as it could be expected, for smaller sets of robots. This is due to the fact that each agent needs to learn how to avoid breaking each constraint on its own state-action space. Still, the results clearly show that the system converges to an optimal solution, no matter the number of robots used.

4.6.2 Uncooperative system

With respect to the cooperative agents system, the only significant difference was the definition of the goal state: each agent i had its own goal position g_i and

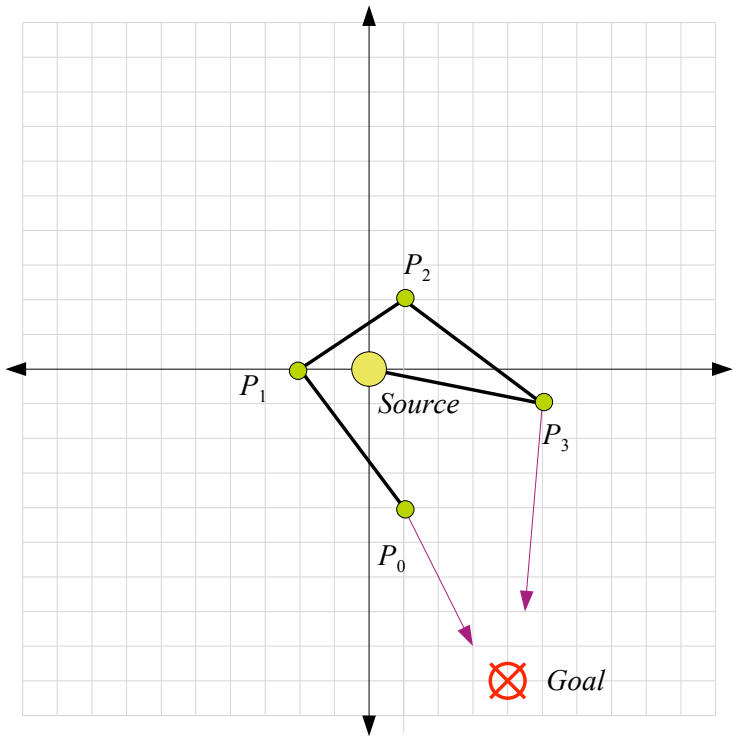
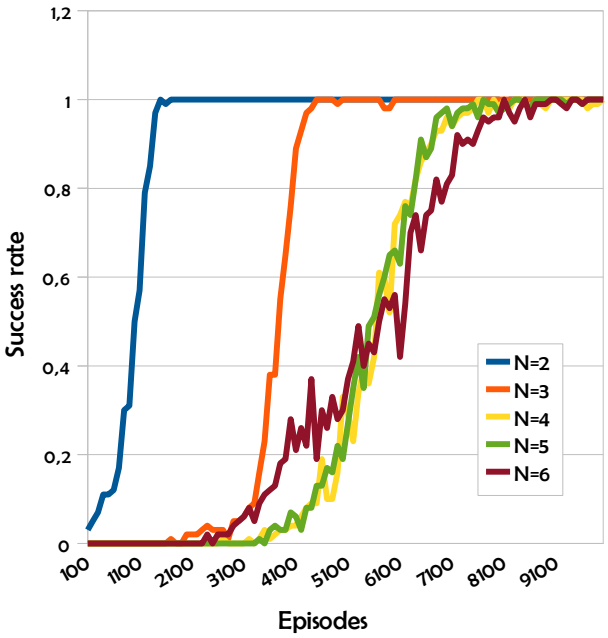
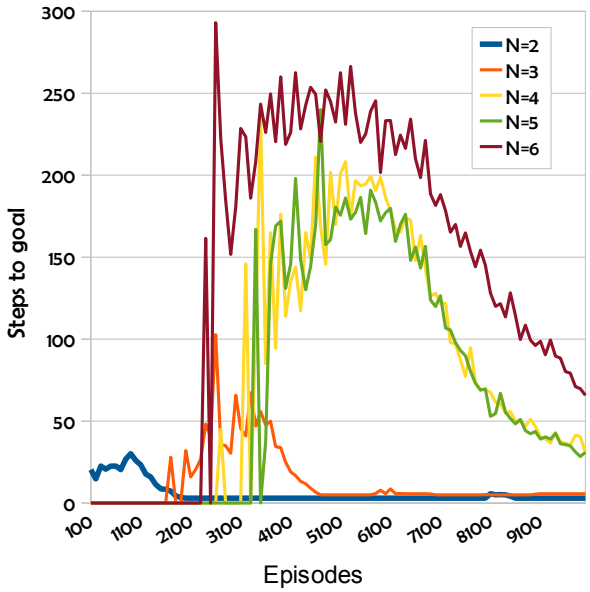


Figure 4.3: Cooperative learning. Visual representation of an instance of the problems due to a lack of cooperation.



(a) Average success rates in the learning process for different configurations ($N = 2, 3, 4, 5, 6$).



(b) Average number of steps needed to reach the goal for different configurations ($N = 2, 3, 4, 5, 6$).

Figure 4.4: Cooperative learning of the simplified hose transportation task: Experiment B.

Goal-modules received a positive reward when the robot controlled by agent i reaches its goal position g_i.

4.6.2.1 Experiment A

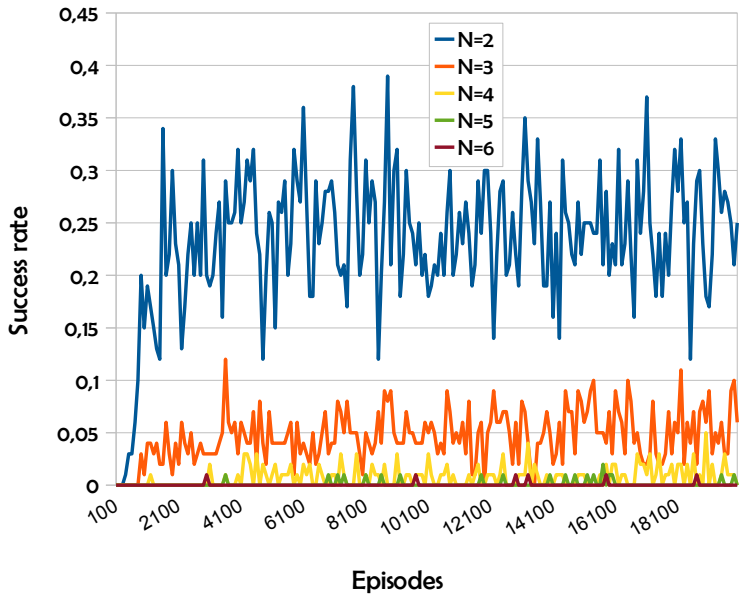
Results show the improvement introduced by the use of the MSAV versus the conventional $\epsilon - greedy$ action selection in the concurrent training of the agents. Videos were generated for each of the experiments to visually validate the simulations. Figure 4.5 shows the evolution of the percentage of episodes reaching the target as the training evolves for both experiments for an increasing number of agents. In both cases, increasing the number of agents dramatically reduces the capacity to reach the goal. However results are much better for the MSAV system (figure 4.5b) than for the baseline $\epsilon - greedy$ system (figure 4.5a) that has very small success rates. Results for the Threshold MSAV are paradoxical. The configurations with the greater number of agents have a slow start of increasing success ratio, but they end achieving much better success rates than the smaller systems, which saturate after a fast increase in success rate. This result is counter-intuitive, because we expect the larger systems to be stuck much easily due to their complexity.

Visual inspection of the videos generated from unsuccessful episodes at the final phases of the learning process show that the biggest difficulty (besides the constraints) for a successful learning of the task is the local nature of the rewards, favoring selfish behaviors. Figure 4.6 shows one instance of the kind of conflicts that may arise. The agent controlling robot P_2 has already reached its goal position and thus, cannot improve its accumulated rewards by taking any action different than *None*. This impedes robots P_0 and P_1 to reach their goal positions. Despite these strong handicaps, the MSAV approach reaches a considerable success as discussed above.

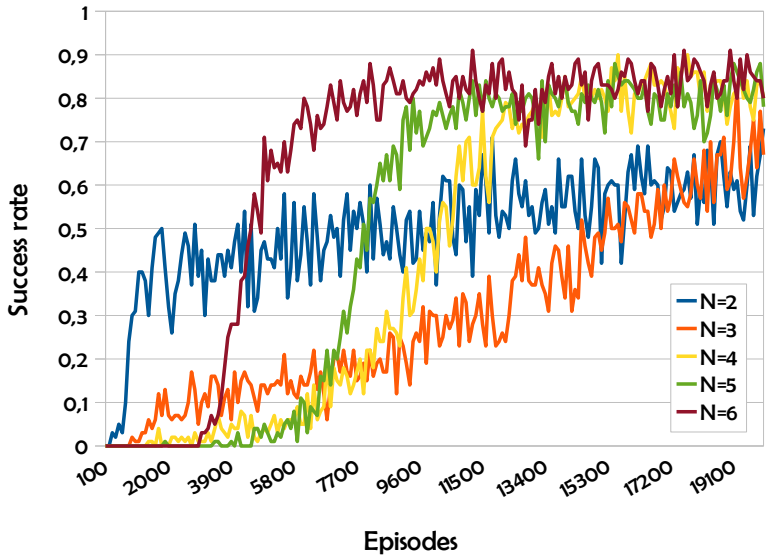
4.7 Conclusions

We have presented in this chapter three novel techniques and studied their appropriateness to learn a decentralized control algorithm for a paradigmatic instance of L-MCRS. First, we introduced the RR-POMDP framework, which we decomposed as a Modular RR-POMDP. Then, we proposed a modular learning algorithm, and finally we introduced Thresholded-MSAV. Different experiments were conducted using these techniques to the multi-robot hose transport problem. Results show that combined use of separate constraint modules and a veto system improves the otherwise very slow monolithic learning rate. We also studied the scalability of the system and, although increasing the number of agents decreased slightly the performance of the learning algorithm, results were good. Another interesting result is that, despite the subset of state variables each agent was able to observe, the system could provide a valid decentralized control algorithm for most of the configurations.

Visual observation of the failed episodes identified two main reasons why the



(a) Success rates for different number of robots without Thresholded MSAV.



(b) Success rate for different number of robots using Thresholded MSAV.

Figure 4.5: Non-cooperative learning of the simplified hose transportation task: experiment A.

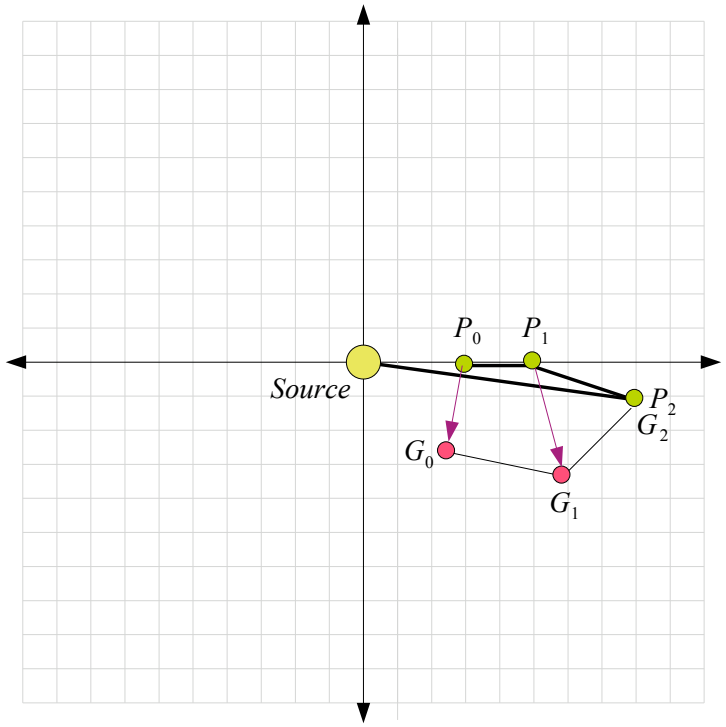


Figure 4.6: Visual representation of the coordination problem (b).

system was not able to deal successfully with the most complicated configurations: (a) the lack of a common perception of the state space and (b) the need of a shared reward signal. Because of the former, big groups of robots are not able to deal with the hose when the hose gets rolled around the source point, and the system has to rely on the random actions selected with probability ϵ to get unstuck. The latter gave place to greedy uncooperative situations in which robots would get blocked by other agents' decisions, and these agents had no incentive to help. Therefore, cooperative systems seem like the most promising approach for L-MCRS.

Chapter 5

Safe MSAV for MDP

In this chapter, we propose Safe Modular State-Action Vetoes (Safe-MSAV), which reduce the storage and computational requirements of the Thresholded vetoes. This approach is based on the fact that optimal policies will never lead the agent to an undesired termination state (UTS), i.e. a broken constraint in a L-MCRS. Therefore, state-action pairs leading to UTS can be identified and vetoed from the first time encountered. Computational experiments show that the use of Safe-MSAV in the exploration process introduces a great speedup of learning convergence.

This chapter is structured as follows. Section 5.1 gives some introductory definitions. Section 5.2 introduces Modular MDPs. Section 5.3 discusses Safe-MSAV policies and studies their convergence properties. We present a simple state variable criterion to learn the correct safe policies faster in Section 5.4. Experimental results comparing Safe-MSAV and standard Q-Learning in two different scenarios are reported in Section 5.5. Finally, conclusions are given in Section 5.6.

5.1 Introduction

We propose the Safe Modular Action-State Vetoes (Safe-MSAV) approach to boost the efficiency of state space exploration, in environments modeled by MDPs with a large number of *undesirable termination states* (UTS). The approach aims to avoid as early as possible state-action pairs that could lead to an UTS using no *a priori* knowledge.

In RL, rewards provide the value of the states in which they are received. Often in practical applications, delayed rewards are used, which means that agents receive null rewards, until some relevant terminal state is reached. We can say that the reward function partitions the state space S into three disjoint subspaces: goal states G, transition states T, and undesirable terminal states (UTS) U, such that $G \cup U \cup T = S$ and $(G \cap U) = (G \cap T) = (U \cap T) = \emptyset$. Formally, these subspaces are defined as follows:

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

All states $s \in G \cup U$ are terminal states which force the learning episode to restart. The goal of the agent is to learn the optimal policy for every state in S. Q-Learning estimates the state-action value repeatedly visiting and updating each state-action pair.

Safe exploration was proposed by Geibel et Al. [46, 44, 45] to compute the risk of reaching an UTS in U. Here we are interested in safe policies, allowing safe and fast exploration of the state-action space. Because optimal policies for states in T cannot make the agent transition to U, imposing vetoes to these transitions the exploration can be expected to be more efficient.

Definition 5. We say that a policy π over a MDP $< S, A, P, R >$ is *U-safe* if the probability of taking an action leading to a state $s \in U$ is 0. Formally:

$$\forall s \in T; \forall s' \in U; \forall a \in A;\ P(s, a, s') > 0 \Rightarrow \pi(s, a) = 0, \tag{5.1}$$

Our goal is to learn a U-Safe policy as fast as possible. Our hypothesis is that this will help the agent learn the optimal values of states in T faster.

5.2 Modular MDPs

We have tackled the learning of U-Safe policies with a modular decomposition of the Monolithic MDP in which several Veto-modules learn which state-action pairs imply a risk of transition into U. Notice that Modular MDP are single-agent systems that give a computationally efficient representation.

Definition 6. A Modular MDP $< S_i, A, P, R_i >^m$, $i = 1, \ldots, m$ is a set of m concurrent independent MDPs with disjoint state subspaces, different reward functions, and a common action space.

Assuming the following decomposition of the state space $S = S_X \times S_Y$, S_X and S_Y are respectively the subspaces spanned by state variables $X = \{x_1, x_2, x_3 \ldots\}$ and $Y = \{y_1, y_2, y_3, \ldots\}$. We denote $[s]_{S_X}$ the projection of a state $s \in S$ into subspace S_X, spanned by state variables in X. The expression $s' \models [s]_{S_X}$ means that $s' \in S_X$ agrees with $s \in S$ on every state variable in X.

Definition 7. A Modular MDP $< S_i, A, P, R_i >^m$ is a decomposition of a Monolithic MDP $< S, A, P, R >$ if $S = \times_{i=1}^{m} S_i$, and $\forall s \in S;\ R(s) = \sum_{i=1}^{m} R_i(s)$.

The Safe-MSAV approach considers different reward signals, each fed to a different module inside the agent, as represented in Figure 5.1. The Safe-MSAV Modular MDP is composed of two types of modules: one *Goal Module* <

$S^G, A, P, R^G >$, which learns the main task, and $m-1$ different *Veto Modules* $< S_i^U, A, P, R_i^U >$ each corresponding to a class of UTS. Each module observes *only the relevant subset of state variables* and receives a specific reward signal. The Goal Module state variables are denoted X^G spanning its local state subspace S^G. Each of the $m-1$ Veto Modules has state variables denoted $X_i^U, i, \ldots, m-1$ spanning local state subspaces S_i^U, $i = 1, \ldots, m-1$. The complete set of state variables observable by the agent is denoted $X = \left\{X^G, X_1^U, \ldots, X_{m-1}^U\right\}$, spanning the complete state space $S = S^G \times \left(\times_{i=1}^{m-1} S_i^U\right)$. The reward signals $R^G(s)$ and $R_i^U(s), i = 1, \ldots, m-1$ are fed to the Goal Module and Veto Modules, respectively.

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

Because of the state space partition $G \cap U = \emptyset$, it is safe to assume that no transition (s, a, s') can trigger both $R^G(s)$ and some $R_i^U(s)$. Therefore we can formulate the agent's perceived reward as

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

which will be positive only when the system reaches the goal state, and negative when some UTS is reached and the corresponding alarm is raised.

The linear decomposition of rewards is supported in a further partition of the undesired terminal states $U_i = \left\{s \mid s \in S,\ R_i^U(s) < 0\right\}$, such that $\forall i, j = 1, \ldots, m-1;\ i \neq j \Rightarrow U_i \cap U_j = \emptyset$, and $U = \cup_{i=1}^{m-1} U_i$. Undesired terminal states trigger a unique reward signal, and each reward depends on a disjoint subset of state variables. This means that each module can learn independently and concurrently to avoid states in its corresponding U_i, with a large saving in time.

5.3 Safe-MSAV policies

Safe-MSAV consist on one Goal Module which estimates the state-action value matrix $Q^G\left(s^G, a\right)$ and several Veto-Modules which learn which actions can be safely taken in each state.

Safe action repertoires The *safe action repertoire* for each state s is defined as the set of actions $A^e(s)$ that cannot result in a transition to an undesirable state in U. From the perspective of Veto Module i, the safe action repertoire A_i^e is defined in its own state subspace as:

$$A_i^e\left(s_i^U\right) = \left\{ a \,\middle|\, a \in A \wedge \sum_{s' \in [U]_{S_i^U}} P_i\left(s_i^U, a, s'\right) = 0 \right\},$$

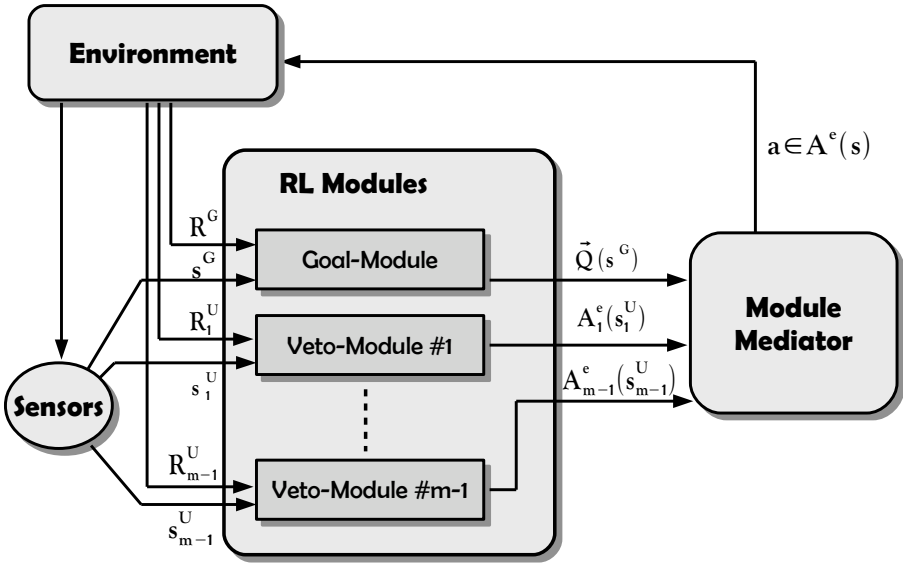


Figure 5.1: Scheme of the Safe-MSAV.

where $s_i^U \in S_i^U$. This set of safe actions can be learnt from experience using the following procedure. Each Veto-Module stores a matrix of boolean values $B_i\left(s_i^U, a\right) = \{true,\ false\}$, $s_i^U \in S_i^U$. This matrix is initially set to $false$, and $B_i\left(s_i^U, a\right) = true$ every time a negative reward $R_i^U\left(s'\right)$ is triggered after a state transition (s, a, s') such that $[s]_{S_i^U} = s_i^U$. Using matrix B the module's safe action repertoire can be computed as

$$A_i^e\left(s_i^U\right) = \left\{a \,|a \in A \wedge \neg B_i\left(s_i^U, a\right)\right\}.$$

Given local safe action repertoires A_i^e, the Module Mediator's performs action selection on the set

$$A^e\left(s\right) = \bigcap_{i=1\ldots m} A_i^e\left(\left[s\right]_{S_i^U}\right).$$

To boost convergence to a *U-Safe* policy (Equation 5.1), Safe-MSAV systems use Veto-Based action selection policies, allowing modules to forbid state-action pairs that have received a negative reward in the past. The existence of a veto can be formalized as a binary valued decision function such as

$$Veto(s, a) = \begin{cases} true & if\ a \notin A^e\left(s\right) \\ false & \text{otherwise} \end{cases},$$

where $s \in S$ is the state. We can now formulate a specific $\epsilon - greedy$ algorithm as:

$$\pi'_{\epsilon}(s,a) = \begin{cases} 0 & a \notin A^e(s) \\ \frac{\epsilon}{|A^e(s)|-1} & (a \in A^e(s)) \wedge a \neq \underset{a' \notin A^e(s)}{\arg\max} \left\{ Q^G\left([s]_{S^G}, a'\right) \right\} \\ 1-\epsilon & (a \in A^e(s)) \wedge a = \underset{a' \notin A^e(s)}{\arg\max} \left\{ Q^G\left([s]_{S^G}, a'\right) \right\} \end{cases}. \quad (5.2)$$

Meaning that if under state s any Veto Module imposes a veto for action a, then $\pi(s,a,\epsilon) = 0$.

Proposition 1. *Let* $< S, A, P, R >$ *be a Monolithic MDP decomposed and trained as a Safe-MSAV Modular MDP* $\{< S^G, A, P, R^G >, < S_i^U, A, P, R_i^U >, i = 1, \ldots, m\}$. *If all state-action pairs are infinitely visited during Q-Learning, applying a Veto-based* $\epsilon - greedy$ *action selection algorithm, the resulting policy will be* U-Safe *for* $< S, A^e(s), P, R >$.

Proof. By construction of the Safe-MSAV Modular MDP, reward signals are linearly composed to give the agent's perceived reward $R(s) = R^G(s) + \sum_{i=1}^{m-1} R_i^U(s)$, where $R^G(s) \geq 0$ and $R_i^U(s) \leq 0$. Therefore, for all undesired termination states $s \in U$ such that $R(s) < 0$, some local reward $R_i^U(s) < 0$ is triggered. Furthermore, for each transition (s, a, s') that receives a negative reward $R(s') < 0$ some veto module i will receive a negative reward $R_i^U(s') < 0$ after a local state transition $\left([s]_{S_i^U}, a, [s']_{S_i^U}\right)$ and the state-action pair (s,a) will eventually be vetoed after observing it. If infinite exploration of state-action pairs is guaranteed for all state-action pairs (the usual requirement for Q-Learning to converge), all pairs (s,a) such that $\sum_{s' \in U} P_i(s,a,s') > 0$ will eventually be vetoed. Given Equation (5.2), the policies learned by the Safe-MSAV Modular MDP and the original Monolithic MDP will be equal:

$$\forall a \epsilon A^e(s); \forall s \epsilon S;\ \pi'_{\epsilon}(s,a,\epsilon) = \pi_{\epsilon}(s,a,\epsilon).$$

Thus, the Safe-MSAV Modular MDP $\{< S^G, A, P, R^G >, < S_1^U, A, P, R_1^U >, \ldots, < S_m^U, A, P, R_m^U >\}$ is equivalent to a Monolithic MDP $< S, A^e(s), P, R >$, inheriting the convergence properties from the original Q-Learning algorithm [122]. □

Our experiments show that RL over the Safe-MSAV Modular MDP $< S, A^e(s), P, R >$ converges to an optimal policy much faster than the original Q-Learning for the Monolithic MDP $< S, A, P, R >$. This improvement is due to the fact that each module uses a subset of the state variables. Our experiments show that the improvement on the exploration directly depends on the state space projection done on each *Veto-Module*. As a general rule, it is usually better to include additional sensory information that can help identify faster the different classes of undesirable termination states. As an example, we have found very effective to use relative positions (which can be provided by some artificial vision sensor)

to determine if distances between robots are within some range, while absolute positions (which can be acquired using some GPS system) are generally best suited to learn to keep a robot inside the simulation space grid.

5.4 Modular state variable relevance

In this section we give a basic algorithm to determine whether a state variable is relevant for a module or not. First, we give a criterion to determine which state variables are irrelevant for a given module.

Definition 8. Let be S the set of states spanned by the values taken by state variables $X \cup Y$, such that $X = \{x_1, x_2, \ldots, x_{k_1}\}$ and $Y = \{y_1, y_2, \ldots, y_{k_2}\}$. We say state variables in Y are *irrelevant for module* i if

$$\forall s \in S;\ \forall s' \models [s]_{S_X}\ ;\ R_i(s,a) = R_i(s',a)\,.$$

To determine which state variables are irrelevant for each module, the agent needs to store experience tuples $\Gamma =< s, a, R_1^U(s,a), \ldots, R_{m-1}^U(s,a) >$, analyzing the available data after a sufficient amount of tuples are available. Algorithm 5.1 determines the relevance of each state variable, and outputs a relevance boolean-matrix $relevant(i,j)$, $i = 1 \ldots, k$, $j = 1, \ldots, |X|$. After executing this procedure, the agent needs only feed each Veto-module j with those state variables x_i such that $relevant(i,j)$ is true. This method offers an straightforward speed-up in the process of learning the constraints. Reward signals $R_j^U(s,a)$ are independent of the irrelevant state variables x_i. This means that as soon as one pair (s,a) is visited and vetoed by module j, all states $s' \in S$ such that $s' \models [s]_{S_j^U}$ may also be vetoed. This implies an obvious speed gain toward the learning of correct safe actions repertoires $A^e(s)$.

In some cases, some state variables will be a linear function of others and, in those cases, depending on the order their relevance is tested, the algorithm will output a different set of relevant variables. If we assume that the observed values for each of them is a representative sample of the range of values they can take, then we would rather test first the relevance of those whose range is bigger. The reason is that, the bigger the range of a removed state variable x_i is, the bigger the number of states that will be projected to $X - \{x_i\}$ is, and thus the faster the convergence to $A^e(s)$. Because of this, Algorithm 5.1 first sorts the state variables $sort(X)$ by the observed range of values each state variable has. The algorithm ends when only one state variable is left in X_j^U or all state variables have already been tested.

Because this algorithm is deterministic and relies completely on the available data, it is only adequate in deterministic and noise-free environments. Stochastic and noisy measurements of the data would require more complex statistical analysis, but this is out of the scope of this work.

Example We will illustrate the idea with a simple example. Consider the maze represented in Figure 5.2. The state space is defined using three different

Algorithm 5.1 Basic algorithm to determine the relevance of state variables for each module.

repeat for each $j = 1 \dots m-1$
 $X_j^U \leftarrow sort\,(X)$
 repeat for each x_i in X_j^U
  $relevant\,(i,j) \leftarrow false$
  $X' \leftarrow X_j^U - \{x_i\}$
  repeat for each (s,a) in Γ
   repeat for each $s' \models [s]_{S_{X'}}$ in Γ
    if $R_j^U\,(s,a) \neq R_j^U\,(s',a)$ then
     $relevant\,(i,j) \leftarrow true$
    end if
   end
  end
  if $\neg relevant\,(i,j)$ then
   $X_j^U \leftarrow X_j^U - \{x_i\}$
  end if
 end
end for

state variables $X = \{x, y, o^{up}\}$, where x is the x-coordinate of the cell occupied by a given robot and y represents its y-coordinate. o^{up} is a boolean flag indicating whether an obstacle is detected in the upper cell. The agent is available to select a unique action: up, which moves the robot up one cell. The reward signal $R\,(s,a)$ is defined for each state as $-k$ if the robot collides with the wall after taking action a in state s, and as 0 otherwise. Table 5.1a represents an example of the experience tuples Γ an agent would gather in such an environment.

Algorithm 5.1 tests first the relevance of state variable x. There are three pairs of instances which share the values of variables in $X - \{x\}$: instance pairs $(0,2)$, $(1,3)$, and $(4,5)$, all of which have observed the same reward. Thus, the algorithm would mark x as irrelevant and remove it from the subset X_j^U. The result is a projection into a reduced number of state variables, and it is shown in Table 5.1b. The algorithm would then proceed with variable y: the same three instance pairs $(0,2)$, $(1,3)$ and $(4,5)$ share the same values in state variables $X - \{x,y\}$ and the same rewards. The algorithm would mark y as irrelevant for $R\,(s,a)$ and would end (only one state variable is left), determining that o^{up} is the only relevant state variable.

5.5 Experimental results

To measure the convergence speed-up due to Safe-MSAV policies, we approached two different single-agent tasks: the hose transportation using a simplified model, and a single-agent taxi problem.

instance	x	y	o^{up}	$R(s, up)$
0	0	0	1	$-k$
1	1	0	0	0
2	2	0	1	$-k$
3	3	0	0	0
4	2	1	1	$-k$
5	3	1	1	$-k$

(a) Γ before the state variable relevance test begins. State variables are already sorted by value range from left to right.

instance	y	o^{up}	$R(s, up)$
0	0	1	$-k$
1	0	0	0
2	0	1	$-k$
3	0	0	0
4	1	1	$-k$
5	1	1	$-k$

(b) Γ after the first cycle: x is found irrelevant and discarded.

instance	o^{up}	$R(s, up)$
0	1	$-k$
1	0	0
2	1	$-k$
3	0	0
4	1	$-k$
5	1	$-k$

(c) Γ after state variable y is also discarded

Table 5.1: An example of the proposed modular state variable relevance process.

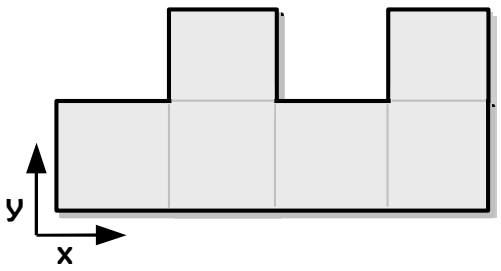


Figure 5.2: State variable relevance example.

5.5.1 Simplified model hose transportation task

We approached the simplified model hose transportation task as a single-agent learning problem with four robots ($N = 4$), where the agent chooses actions from $A = \bigcup_{i=1...N} A_i$, where $A_i = \{Up_i, Down_i, Left_i, Right_i\}$. Experiments were first run using standard Q-Learning on a Monolithic MDP (*Monolithic QL*) with the composite reward signal, and secondly on a Safe-MSAV Modular MDP (*MSAV Modular QL*). One Goal module and four Veto modules were trained independently on their relevant state variables receiving independent local rewards $R_i^{strecth}(s)$, $i = 1, \ldots, N$. In both learning experiments, $n_e = 75000$ episodes were run and Q-learning parameters were set with fixed values $\alpha = 0.25$ and $\gamma = 0.9$. The exploration-exploitation parameter was initially $\epsilon = 1$, decreasing it $\frac{1}{n_e}$ after each episode. All experiments were run using the same pseudo-random sequence of numbers and, for the sake of a fair comparison, invalid state-actions were also vetoed in the Monolithic QL (using the complete representation of the state instead of a subspace). In these experiments, we only used reward signals R_i^g and $R_i^{stretch}$.

After 1000 steps of random exploration, the agent was able to determine which state variables were relevant for each module as proposed in Algorithm 5.1. For $R_i^{strecth}$ reward signals, only the relative positions of the neighbors r_i^n and r_i^p were determined to be relevant.

Figure 5.3 shows the evolution of the average number of steps per episode required by the Safe-MSAV Modular QL to reach the goal using vetoes. As the ϵ parameter is decreased, the system converges to an optimal policy that reaches the goal position in 17 steps. By implementation convention the average number of steps is zero when no goal states where reached during the allowed number of episodes. Figure 5.4a represents the normalized average rate of successful episodes and Figure 5.4b represents the failed episode rate. When ϵ is decreased, as the undesired terminal state-actions are vetoed, the probability of reaching an undesired state also decreases and more episodes end successfully reaching the goal position. Around episode $38000 - 40000$, most of the undesirable state-actions have already been learned.

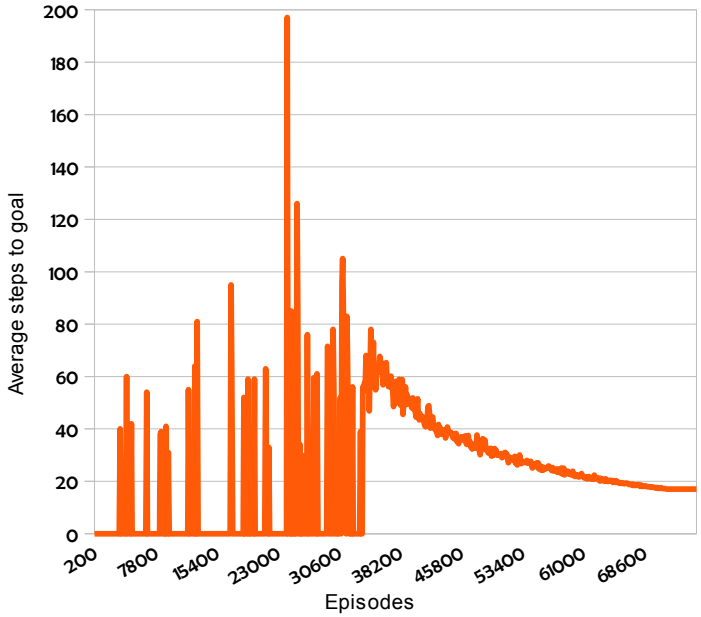


Figure 5.3: Learning results using MSAV Modular QL. Average steps per episode for the tip of the hose to reach the goal position,

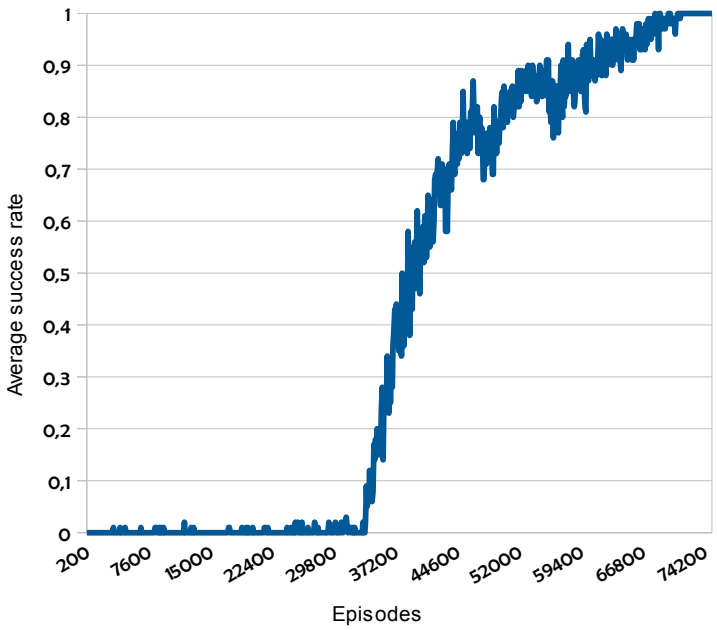
Results for Monolithic QL can not be shown in the graphs for comparison, because the success rate never got above 0.01. This is due to the huge number of state-pairs that result in UTS. Learning them on the complete state variable set is very slow and this justifies the decomposition of the reward signal so each decomposed signal can be learned on different sets of relevant variables.

As an alternative measure of the exploration performance, we represented in Figure 5.5 the number of valid state-action pairs visited during exploration with both algorithms: the higher number of valid pairs visited, the higher the probability of reaching the goal position. The plot clearly shows that the probability of discovering earlier a goal using the MSAV Modular QL is much higher than using a Monolithic QL.

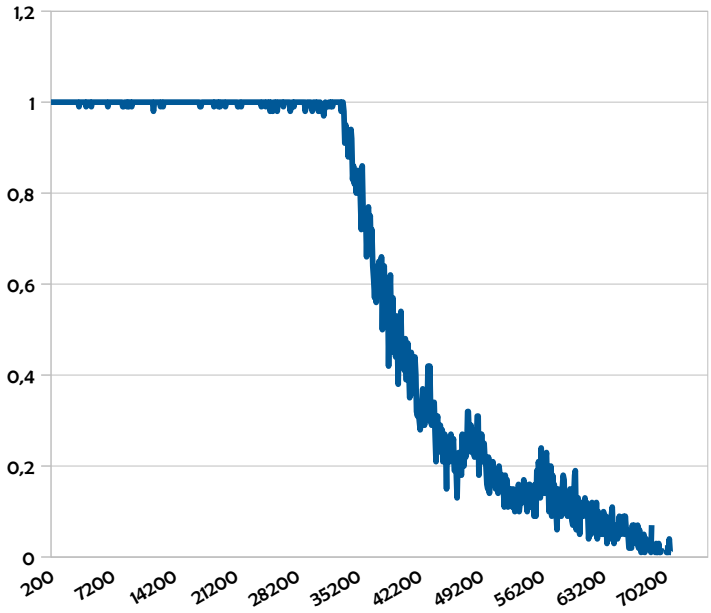
5.5.2 Single-agent taxi domain

The taxi domain was initially proposed as a multi-agent task, but we use it as a single-agent problem to show the effect of Safe-MSAV in the RL process, even when receiving a negative reward does not require to start over the episode. In those cases, we expect that rejecting those state-action pairs that trigger negative rewards contributes to a faster learning of the optimal policy.

The system was trained on the task using standard $\epsilon - greedy$ and the Safe-MSAV $\epsilon - greedy$ policy $\pi'_\epsilon(s,a)$. Initially, $\epsilon = 1$ and, at the end of each



(a) Average success rate.



(b) Failed episode rate.

Figure 5.4: Learning results using Safe-MSAV. Results are averaged using a 100 episode-window.

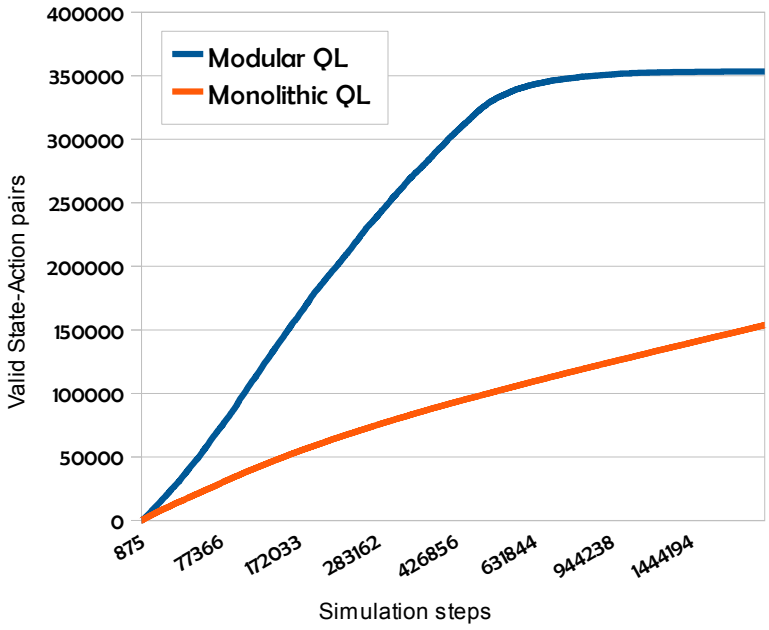


Figure 5.5: Number of valid state-action pairs visited with Monolithic QL (red) and MSAV Modular QL (blue).

episode, a constant $\Delta\epsilon$ was applied to it until $\epsilon = 0$. After only 20 steps of random exploration, the agent was able to determine which state variables were relevant for each module as proposed in Algorithm 5.1. For $R_i^{collision}$ reward signals, state variables $o_i^{up}, o_i^{down}, o_i^{left}$ and o_i^{right} were considered to be relevant.

During the learning process, the quality of the learnt policies were evaluated each 10 episodes greedily selecting the actions with the highest Q-values. This whole learning and evaluation process was repeated for $\Delta\epsilon = 0.002$ and $\Delta\epsilon = 0.005$. Note that the amount of experience the agent is allowed directly depends on this decrement: the lower it is, the higher the allowed amount of time to learn.

Figures 5.6a and 5.7a respectively show the accumulated rewards obtained during the learning process for $\Delta\epsilon = 0.002$ and $\Delta\epsilon = 0.005$. In both cases, the accumulated rewards are higher for the Safe-MSAV policy than those for the standard $\epsilon - greedy$. Figures 5.6b and 5.7b respectively show the number of steps needed by the agents to reach the goal selecting greedily highest Q-values for $\Delta\epsilon = 0.002$ and $\Delta\epsilon = 0.005$. In both cases, the Safe-Safe-MSAV policy is able to learn a better greedy policy. The difference between Safe-MSAV and the standard algorithm are in general best for $\triangle\epsilon = 0.002$.

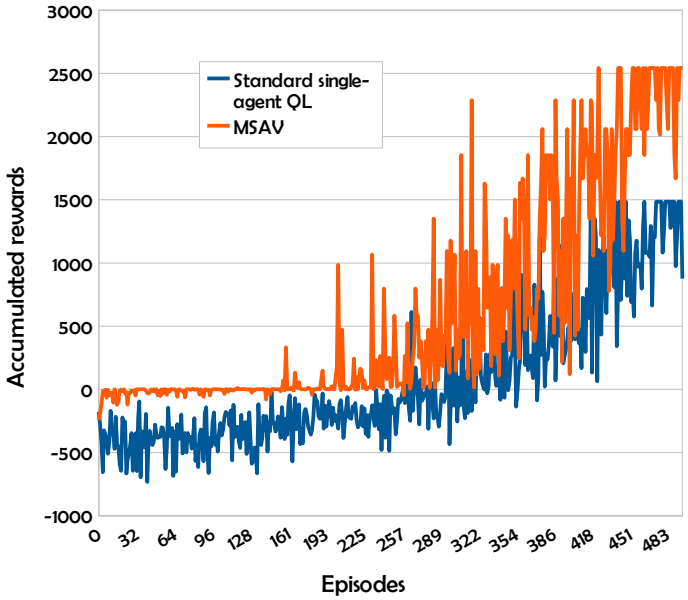
5.6 Conclusions

This chapter presents Safe-MSAV, a novel RL strategy for complex environments with many undesired termination conditions. Its boosted convergence is due to a more efficient exploration of the state-action space. Safe-MSAV uses decomposed reward signals and state subspaces to learn how to detect undesirable state-action pairs faster. Experimental results in a hose transportation task where the number of UTS is unmanageable for the Q-Learning algorithm show the superior convergence properties of the Safe-MSAV. Experiments were also conducted in a single-agent version of the classical multi-agent taxi domain, showing that Safe-MSAV makes a better use of the available learning time. Besides, Safe-MSAV Q-learning allows the concurrent learning of module responses due to the independence between state variables.

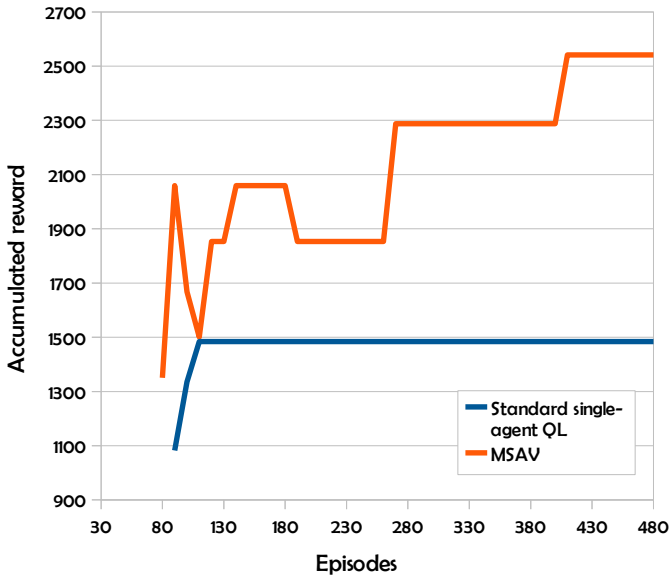
We have also presented a simple algorithm to automatically determine which state variables should be used to model the state in each Veto-module, rather than hand-picking them. Selection of the state variables is performed on a basis of relevance to each Veto-module's reward signal.

In this work, we have applied a tabular implementation of Modular RL, but the approach can be easily extended to the existing functional approximations to the state-action value map: weighted Radial Basis Functions [70], Cerebellar Model Arithmetic Computers [15, 108], Artificial Neural Networks [26, 124], instance-based approximators [3], and so on.

The good results obtained encourage further work on this area, and we believe diverse applications can benefit from the Safe-MSAV approach, mostly environments in which undesirable termination conditions are present.

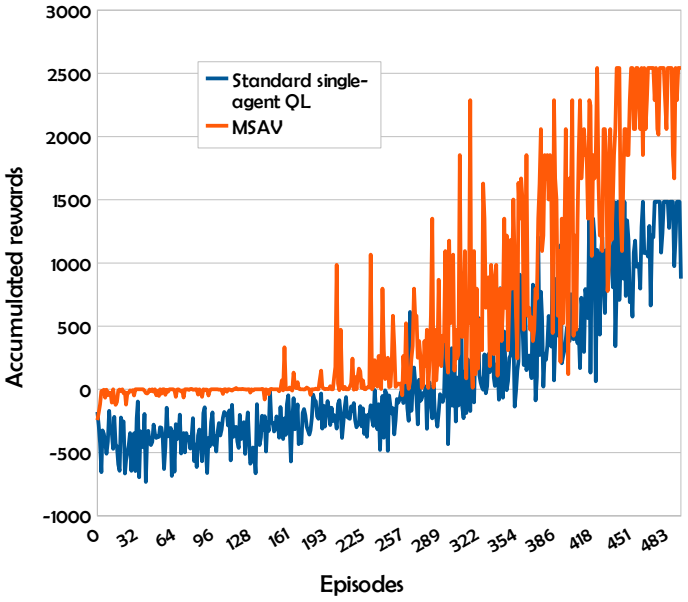


(a) Accumulated rewards during the learning process.

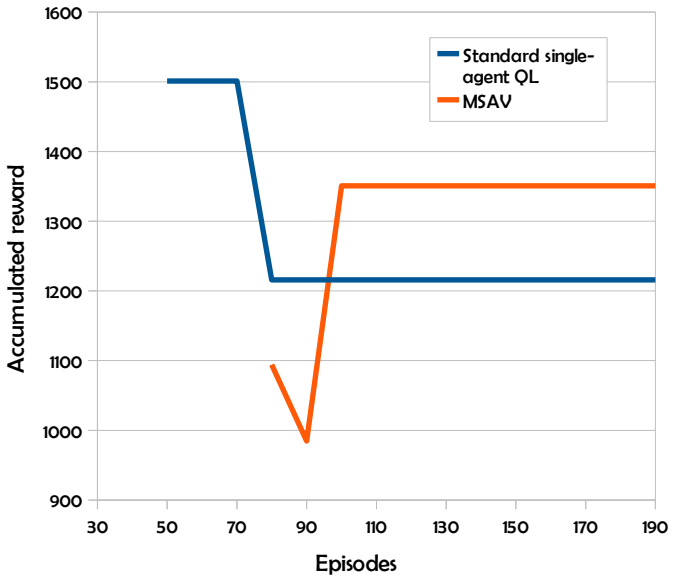


(b) Accumulated rewards obtained greedily selecting learnt Q-values.

Figure 5.6: Learning results for $\Delta\epsilon = 0.002$



(a) Accumulated rewards during the learning process.



(b) Accumulated rewards obtained greedily selecting learnt Q-values.

Figure 5.7: Learning results for $\Delta\epsilon = 0.005$

Chapter 6

Safe MSAV Vetoes for SG

Multi-Agent Reinforcement Learning (MARL) algorithms face two main difficulties: the curse of dimensionality, and the non-stationarity due to the concurrent learning carried out by the agents independently. In this chapter we propose a distributed algorithm that shows linear complexity increase with the number of agents, and reduces the effect of non stationary agent responses by performing a round-robin scheduling of the action selection. We show that this learning scheme has improved convergence properties which hold also in stochastic environments. Moreover, when dealing with over-constrained systems with many undesired termination states (UTS), this MARL algorithm allows the implementation of Modular State-Action Veto (MSAV) which speed up learning convergence by isolating state-action pairs which lead to UTS. Each agent can be trained independently learning the value of local actions, including the MASV policies. Coordination of locally optimal policies is immediate in the round-robin distributed Q-learning. We show that the determination of the turns can be performed in a distributed consensus process and we report our computational results on two different environments.

This chapter is structured as follows. Section 6.1 defines Cooperative Round-Robin Stochastic Games. Section 6.2 proposes a centralized algorithm to learn C-RR-SG. Section 6.3 proposes a distributed communication-free version of the centralized algorithm and Section 6.4 proposes how to compose separately learnt local policies into a joint-policy. Section 6.5 implements C-RR-SG turns using consensus. Section 6.6 reports our computational results and, finally, Section 6.7 offers our conclusions.

6.1 Cooperative Round-Robin Stochastic Games

To reduce the complexity of dealing with joint actions, we transform them into a sequence by considering Round-Robin (RR) task-scheduling distributing turns to execute actions. The sequencing has the effect of removing the non-stationarity introduced by the changing policies of the agents, because all agents but one will be inactive at each turn. The drawback of this approach is that it is not possible to ensure in general that the original Stochastic Game might be realized as a sequential game. The Cooperative Round-Robin Stochastic Game formalizes this idea.

Definition 9. A *Cooperative Round-Robin Stochastic Game* (C-RR-SG) is a tuple $< S, A_1 \dots A_N, P, R, \delta >$, where

- N is the number of agents.
- S is the set of states, fully observable by all the agents.
- A is the global set of actions gathering all local actions: $A = \underset{i=1\dots N}{\cup} A_i$.
- $P : S \times A \times S \to [0,1]$, $i = 1, \dots, N$ is the state transition function $P(s,a,s')$ that defines the probability of observing s' after agent i executes action a from its local action space A_i.
- $R : S \times A \times S \to \mathbb{R}$ is the shared scalar reward signal received after executing a local action from A_i.
- $\delta : \mathbb{N} \to \mathbb{N} \in [1, N]$ is the turn function defining the cycle of action execution. $\delta(t)$ gives the index of the agent allowed to execute an action in time t (times are relative to the start of the episode). This function is cyclic, therefore $\forall t \in \mathbb{N};\ \delta(t) = \delta(t+N)$.

The state value function $V^{\boldsymbol{\pi}}(s,i)$ gives the value of being in state s at turn i under joint policy $\boldsymbol{\pi}$, and it can be expressed as 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

$$
\begin{aligned}
V^{\boldsymbol{\pi}}(s,i) &= E^{\boldsymbol{\pi}}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\} \\
&= E^{\boldsymbol{\pi}}\left\{r_t + \sum_{k=1}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\} \\
&= \sum_{a\in A_i}\pi_i(s,a,i)\sum_{s'}P(s,a,s')\left[R(s,a,s') + \gamma E^{\boldsymbol{\pi}}\left\{\sum_{k=1}^{\infty}\gamma^k r_{t+k+2} \mid s_{t+1} = s'\right\}\right] \\
&= \sum_{a\in A_i}\pi_i(s,a,i)\sum_{s'}P(s,a,s')\left[R(s,a,s') + \gamma V(s',j)\right],
\end{aligned}
$$

Algorithm 6.1 Standard single agent Q-learning of the optimal policy for a C-RR-SG

Initialize $[Q_0(s,a,i)=0; s\in S; a\in A; i=1\dots N]$ arbitrarily
Repeat (for each episode) n:
Repeat (for each step t of episode):

- Observe current state s_t
- Select and execute action a_t
- Observe reward r_t and new state s_{t+1}
- Update the state-action value matrix
  - if $s_t=s$, $a_t=a$, $i=\delta(t)$ and $j=\delta(t+1)$ then

$$Q_t(s,a,i)=(1-\alpha_t)\,Q_{t-1}(s,a,i)+\alpha_t\left[r_t+\gamma\max_{a'}Q_{t-1}(s_{t+1},a',j)\right]$$

  - $Q_t(s,a,i)=Q_{t-1}(s,a,i)$ otherwise

until s_{t+1} is terminal

where j is the index of the agent whose turn follows i as determined by $\delta(t)$, and $E^{\boldsymbol{\pi}}$ represents the expected rewards from time t following joint policy $\boldsymbol{\pi}$. The optimal state value $V^*(s,i)$ is given by:

$$V^*(s,i)=\max_{a\in A_i}\left\{\sum_{s'}P(s,a,s')\left[R(s,a,s')+\gamma V^*(s',j)\right]\right\}. \tag{6.1}$$

The joint state-action value function can be expressed as

$$Q^{\boldsymbol{\pi}}(s,a,i)=\sum_{s'}P(s,a,s')\left[R(s,a,s')+\gamma V^{\boldsymbol{\pi}}(s',j)\right] \tag{6.2}$$

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

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

6.2 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, as illustrated in Algorithm 6.1.

Proposition 2. *Under the same conditions that guarantee convergence of Q-Learning to optimal state-action values in a MDP, centralized single-agent Q-Learning will converge to $Q^*(s,a,i)$ in a C-RR-SG.*

Proof. Single-agent Q-Learning converges to optimal $Q^*(s,a)$ values under two conditions [122]: all state-action pairs (s,a), $a \in A$, $s \in S$ are visited infinite times, and the learning gain α_t is properly decreased according to a classical stochastic convergence theorem. To prove the proposition, it is enough to show that the C-RR-SG can be mapped into a MDP. The turn i can be introduced as an additional state variable whose value is independent of the other variables, and its transitions are determined by $\delta(t)$. Then, the state value function and the state-action value function are identical to the ones of a conventional MDP in this enlarged state space. The Q-table is composed of identical blocks indexed by the turn variable. Besides, in the learning process of Algorithm 6.1, actions are selected on the independent blocks depending only of the turn variable. Therefore, the learning process is an interleaving on N independent Q-learning processes over non interacting MDP, resulting in single Q-learning over a single MDP. Therefore, if the aforementioned two conditions are met, Algorithm 6.1 will converge to local optimal state-action values $Q^*(s,a,i)$. □

Algorithm 6.1 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 a 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 $N \cdot \mid S \times \underset{i=1\ldots N}{\cup} A_i \mid$ entries in its Q-table.

6.3 Distributed Round-Robin Q-Learning for C-RR-SG

We present two different distributed Round-Robin Q-Learning algorithms (D-RR-QL) for C-RR-SG: the first one requires information to be sent between consecutive turn agents each time-step, and a second which is communication-free. In both cases, convergence to an optimal policy is guaranteed. 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 $Q(s,a,i) = Q^i(s,a)$. In the first implementation, the Q-table is updated using the information from the next agent in the RR scheduling cycle, so that the updating does not need to wait the full cycle. However it requires that the j-th agent informs the i-th agent of its optimal policy given by $\max_{a'} Q^j_{t-1}(s_{t+1}, a')$. This algorithm is specified in Algorithm 6.2. Convergence properties are inherited from the centralized single-agent algorithm.

Let us consider the *corrected N-step truncated return* [106]:

$$\Delta V_t(s_t,i) = \alpha \left[R_t^{(N)} - V_t(s_t, \delta(t)) \right], \tag{6.3}$$

Algorithm 6.2 Local estimation of the i-th agent's local Q^i matrix by the message passing D-RR-QL algorithm.

Initialize $\left[Q_0^i(s,a,i)=0; s\in S; a\in A; i=1\dots N\right]$ arbitrarily
Repeat (for each episode) n:
Repeat (for each step t of episode):

- Wait turn
- Observe current state s_t
- Select and execute action a_t
- Observe reward r_t and new state s_{t+1}
- Receive $\max_{a'} Q_{t-1}^j\left(s_{t+1},a'\right)$ from agent $j=\delta\left(t+1\right)$
- Update the state-action value matrix
  - if $s_t=s$, $a_t=a$, $i=\delta\left(t\right)$ and $j=\delta\left(t+1\right)$

$$Q_t^i(s,a)=\left(1-\alpha_t\right)Q_{t-1}^i(s,a)+\alpha_t\left[r_t+\gamma\max_{a'}Q_{t-1}^j\left(s_{t+1},a'\right)\right]$$

  - $Q_t^i(s,a)=Q_{t-1}^i(s,a)$ otherwise
- Give turn, allowing a RR cycle of actions

until s_{t+1} is terminal

Algorithm 6.3 Local estimation of the i-th agent's local Q^i function by the communication-free D-RR-QL algorithm assuming a global time counter visible to all agents.

Initialize $\left[Q_0^i(s,a,i)=0; s\in S; a\in A; i=1\dots N\right]$ arbitrarily
Repeat (for each episode) n:
Repeat (for each step t of episode):

- Wait turn
- Observe current state s_t
- Select and execute action a_t
- Observe the rewards $r_t,\dots,r_{t+N-1}$ of a complete RR cycle and new state s_{t+N} after the RR cycle.
- Learn the state-action value matrix
  - if $s_t=s$, $a_t=a$, $i=\delta(t)$ and $j=\delta(j)$

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

  - $Q_t^i(s,a)=Q_{t-1}^i(s,a)$ otherwise

until s_{t+N} is terminal

where $V_t(s_t, \delta(t))$ is the value of being in state s_t at time t, and $R_t^{(N)} = \sum_{k=0}^{N-1} \gamma^k r_{t+k+1} + \gamma^N V_t(s_{t+N}, \delta(t))$. Equation 6.3 is the error that we commit if we wait N-steps for the estimation of the state value. From this expression, we can derive an update rule for the state-action Q-values using the sequence of N rewards observed during a RR cycle starting from agent i and the state s_{t+N} observed at the end of the cycle:

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

This update rule is very convenient for Q-learning in a C-RR-SG because each agent can update its local Q-table without needing to know the Q-tables of other agents. The rule:

$$Q_t^i(s,a) = (1-\alpha_t) Q_{t-N}^i(s,a) + \alpha_t \left[\sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^N \max_{a'} Q_t^i(s_{t+N}, a') \right] \quad (6.4)$$

is applied when $s_t = s, a_t = a, \delta(t) = \delta(t-N) = i$ which is used in the D-RR-QL of Algorithm 6.3 to learn the optimal policies for a given C-RR-SG, updating them at the end of the execution cycle. This is the communication-free implementation of the D-RR-QL algorithm which will be the subject of our experiments.

Proposition 3. *Convergence of the D-RR-QL Algorithm 6.3 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 $< S, A_1 \ldots A_N, P, R, \delta >$ is guaranteed under the equivalent conditions required for single-agent Q-Learning to converge in a MDP.*

Proof. Sutton et al.[106] showed that N-step TD Methods converge to optimal value functions $V^*(s)$ due to their *error reduction property* [121]:

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

which means that the worst error for the one-step ahead estimate $V_t(s_t)$ gives an upper bound for the error committed with the N-step ahead estimation $E^{\boldsymbol{\pi}} \left\{ R_t^{(N)} \mid s_t = s \right\}$. Besides, this error bound can be made as tight as desired because the factor γ^N goes to zero as N grows. Therefore, each agent in the D-RR-QL of Algorithm 6.3 can be interpreted as performing a N-step ahead updating of the value functions, whose error relative to a single agent Q-learning on a MDP is bounded and as small as required. Considered as independent processes, the addition of rewards during a RR cycle for each agent are independent

variables, therefore the local updating of $Q^i(s,a)$ are independent interleaved in time Q-learning processes, which converge to the optimal $Q^*(s,a,i)$ if the stochastic gradient conditions of [122] hold. □

6.4 Composition of concurrent joint-policies

A C-RR-SG can be considered an approximation to the original C-SG problem, where only disjoint actions following a predefined order can be taken. Policies learned using this approximation are very likely to be suboptimal in the original SG setting, because the discounted reward approach assumes task completion instant to be critical in 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 C-SG where actions can be executed concurrently in any order. Therefore, we deal with a combinatorial optimization problem. We present a coordination algorithm to build an approximation of the optimal policy $\boldsymbol{\pi}(s,\mathbf{a})$ for a C-SG from the distributed Q-tables learned using D-RR-QL.

Definition 10. A *C-RR-SG* $< S, A_1 \dots A_N, P, R, \delta >$ is a *sequential realization* of a *C-SG* $< S, A_1 \dots A_N, \mathbf{P}, \mathbf{R} >$[1] if the following two properties are met:

- $\forall s, s', \mathbf{a}, i;\ \mathbf{P}(s, \mathbf{a}_i, s') = P(s, a_i, s')$
- $\forall s, s', \mathbf{a}, i;\ \mathbf{R}(s, \mathbf{a}_i, s') = R(s, a_i, s')$,

where a joint-action $\mathbf{a}_i$ is defined as a joint-action where only the i-th agent has a non null action, i.e. $\mathbf{a}_i = [\emptyset, \dots, \emptyset, a_i, \emptyset, \dots, \emptyset]$.

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

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

where $P\left(s, a_{\delta(1)}, a_{\delta(2)} \dots a_{\delta(N)}, s'\right)$ and $R\left(s, a_{\delta(1)}, a_{\delta(2)} \dots a_{\delta(N)}, s'\right)$ represent the probability of reaching state s' after executing actions 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

[1] For clarity, we will use throughout this paper regular characters for local actions (i.e. a, a_i, A) and bold characters for joint-actions (i.e. $\mathbf{a}$, $\mathbf{a}_i$, $\mathbf{A}$). Likewise, we will use regular characters to distinguish those functions defined for joint-actions ($\mathbf{Q}$, $\mathbf{P}$ and $\mathbf{R}$) from those defined for local actions (Q, P and R).

Figure 6.1: Basic example to illustrate the assumptions made for the joint-action composition procedure. Two robots ($R1$ and $R2$) and the goal position in blue.

$$\mathbf{Q^{\pi}}(s_0, \mathbf{a}) = \sum_{s_1 \ldots s_N} \left(\prod_{i=0}^{N-1} P(s_i, a_{i+1}, s_{i+1}) \right) \tag{6.5}$$

$$\cdot \left\{ \sum_{i=0}^{N-1} \gamma^i \cdot R(s_i, a_{i+1}, s_{i+1}) + \gamma^N \cdot \max_{a'} Q^{\pi}(s_N, a', \delta(1)) \right\} \tag{6.6}$$

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.

Example 1. For illustration purposes consider the domain represented in Figure 6.1: two robots are located in opposite sides of a 3-cell small room. Action set is the same for both $A = \{left, right, none\}$. The center blue square is the goal position If any robot reaches the goal position, both receive a positive reward (100). If any robot hits a wall or the other robot, both receive a negative reward. Otherwise, they receive a neutral reward. Tables 6.1 and 6.2 represent the state transitions from the initial configuration, and received rewards for all possible combinations of action in concurrent joint-actions (Table 6.1) and sequential individual actions (a_1, a_2) in a RR scheduling $\delta(1) = 1, \delta(2) = 2, \delta(3) = 1, \ldots$ (Table 6.2). Note that if the discount γ is close to 1, the expected rewards are the same, whether actions are taken concurrently or sequentially. The two sequences of actions with maximum return in Table 6.2 ($\{right, none\}$ and $\{none, left\}$) also maximize the expected reward of the respective joint-action in Table 6.1. Almost all state transitions are the same for the concurrent and sequential action execution. For this specific RR turn order δ, the only exception is the transition $\{right, left\}$.

Value Maximization using Variable Elimination The combinatorial optimization problem has been tackled as a greedy variable elimination process [53] 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. 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. Each 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 estimation can be defined as

a_1	a_2	s'after $\mathbf{a} = [a_1 a_2]$	$\mathbf{R}(s, \mathbf{a}, s')$
$left$	$left$	$\{R1, R2, \perp\}$	90
$left$	$right$	$\{R1, R2, \perp\}$	-20
$left$	$none$	$\{R1, R2, \perp\}$	-10
$right$	$left$	$\{R1, R2, \perp\}$ or $\{\perp, R1, R2\}$	90 in either case
$right$	$right$	$\{\perp, R1, R2\}$	90
$right$	$none$	$\{\perp, R1, R2\}$	100
$none$	$left$	$\{R1, R2, \perp\}$	100
$none$	$right$	$\{R1, \perp, R2\}$	-10
$none$	$none$	$\{R1, \perp, R2\}$	0

Table 6.1: Joint-action composition procedure: State transitions and rewards for concurrent joint-actions.

a_1	a_2	s' after a_1, a_2	$R(s, a_1, a_2, s')$
$left$	$left$	$\{R1, R2, \perp\}$	$-10 + \gamma 100$
$left$	$right$	$\{R1, R2, \perp\}$	$-10 - \gamma 10$
$left$	$none$	$\{R1, R2, \perp\}$	-10
$right$	$left$	$\{\perp, R1, R2\}$	$100 - \gamma 10$
$right$	$right$	$\{\perp, R1, R2\}$	$100 - \gamma 10$
$right$	$none$	$\{\perp, R1, R2\}$	100
$none$	$left$	$\{R1, R2, \perp\}$	100
$none$	$right$	$\{R1, \perp, R2\}$	$-\gamma 10$
$none$	$none$	$\{R1, \perp, R2\}$	0

Table 6.2: Joint-action composition procedure: State transitions and rewards for sequential execution local actions in RR scheduling.

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

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 best policy according to equation (6.5) follows a message-passing scheme. Each agent selects its optimal action according to the local Q-table, then it forwards to the next agent in the RR scheduling the list of state transitions with positive probability when executing the selected action. This message-based procedure is depicted in Figure 6.2.

Starting in a given state s_0, the agent $\delta(1)$ selects $a_1 = \arg\max_a \left\{Q^{\delta(1)}(s_0, a)\right\}$ and sends a message $\left\langle s_1^j, \hat{P}_{\delta(1)}\left(s_0, a_1, s_1^j\right); j = 1, \ldots, k_1\right\rangle$ to the agent $\delta(2)$, where k_1 is the number of states s_1^j such that $\hat{P}_{\delta(1)}\left(s_0, a_1, s_1^i\right) > 0$. Then, the second agent in the execution cycle selects its optimal action on the basis of the possible states:

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

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

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

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

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

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

and sends message $\left\langle s_i^j, \hat{P}\left(s_0, a_1 \ldots a_i, s_i^j\right); j = 1, \ldots, k_i\right\rangle$ to agent $\delta(i+1)$. When the procedure reaches the last agent, the maximization process yields a joint-action $\mathbf{a} = [a_1, a_2, \ldots, a_N]$ such that

$$\begin{aligned}[a_1 a_2 \ldots a_N] \quad = \quad & \arg\max_{a_1, a_2 \ldots a_N} \left\{ \sum_{j=1\ldots k_N} \hat{P}\left(s_0, a_1 \ldots a_N, s_N^j\right)\right. \\ & \cdot \quad \left.\left(\hat{R}_{\delta(N)}\left(s_N^j\right) + \gamma^N \arg\max_a Q^{\delta(N)}\left(s_N^j, a\right)\right)\right\}.\end{aligned}$$

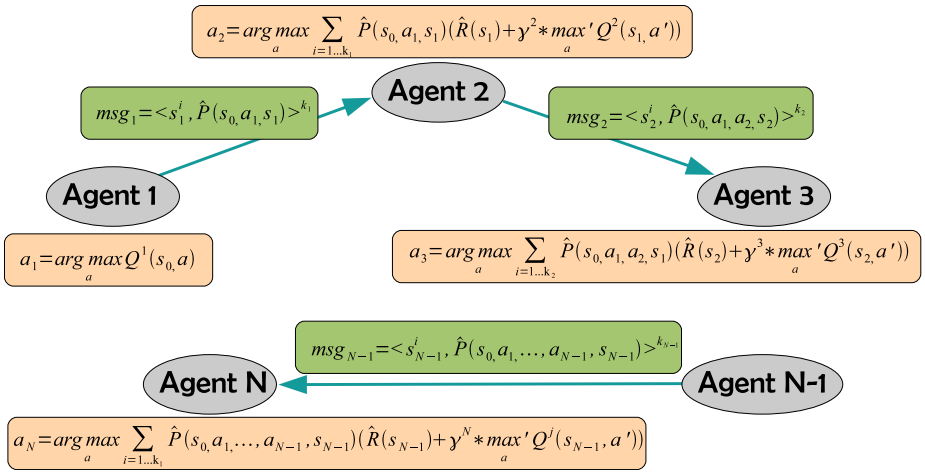


Figure 6.2: Agreement on a joint-action using the message-based procedure.

To give a complete specification, 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 information from this state on. We can address this potential implementation issue by leaving entries in the incoming messages for which no transition probability is available unaltered. The agent will perform no action at all, assuming a terminal state will be reached no matter what the rest of local actions are.

This approximation has experimentally been tested with results showing its validity in environments with delayed rewards. Unlike the technique presented in [53], where the message network could have different topologies, this message passing process we have just presented only needs one cycle to finish. In some cases, it is able to learn an optimal policy and, in some others it is not. Our educated guess is that this mostly depends on the specific RR schedule δ chosen and the degree to which our assumptions are met.

6.5 Consensus-Based Round-Robin Turn scheduling.

A truly distributed implementation of the D-RR-QL avoiding any central controller needs a decentralized protocol to agree on which agent is the owner of the current turn. This section presents a consensus based algorithm with limited communications and low computational requirements. Consensus-Based methodologies have been previously proposed [94] to derive distributed control algorithms using coordination variables [82]. In our context, agents need only agree on the owner of the current turn. Each agent keeps a local estimation ξ_i of the coordination variable $\xi \in \mathbb{R}$ value, which can be updated using a

conventional discrete consensus algorithm:

$$\dot{\xi}_i = -\sum_{j=1}^{N} a_{ij}(t)(\xi_i - \xi_j) + v, \tag{6.7}$$

where $a_{ij}(t)$ coefficients weight the influence of the coordination variable estimations received from other agents, and v is a predefined constant velocity. For simplicity, we assume $a_{ij} = 1/N$. Denoting T the maximum time needed to take an action $a \in \bar{A}$, we define $v = 1/T$. If an action takes less time than T (i.e., null actions), the local instance of the algorithm can update its ξ_i and propagate it to the rest of agents, thus converging to the next turn faster. The coordination variable is in fact the common time counter. Each agent can calculate its estimation of the current time t from its local instance ξ_i as

$$t_i = \lfloor \xi_i \rfloor, \tag{6.8}$$

where the operator $\lfloor . \rfloor$ denotes the greatest integer lower or equal to ξ_i. Agents can then calculate the agent id owning the turn by invoking function $\delta(t_i)$.

Consensus-Based implementation of turns To introduce Consensus-Based turn scheduling in Algorithm 6.3, the pseudo-code step *Wait turn* can be implemented using the local estimation of the turn: *Wait until* $\delta(t_i) = i$. On the other hand, *Give turn* can be accomplished by the increments on the local instance ξ_i: $\xi_i \leftarrow t_i + 1$.

Figure 6.3 represents the complete control scheme, with n agents interacting with the environment and the communication network. Note that rewards $r_1, \ldots, r_n$ should be observed and stored whenever the local variable t_i is updated by the equation (6.7). On the other hand, *Give turn* pseudo-function implementation takes advantage of possibly shorter or even null actions by manually updating the local instance ξ_i instead of waiting for the CA to update it.

6.6 Experiments

We consider two deterministic environment problems to compare the performance of the D-RR-QL with MSAV against the conventional D-QL [72]. Policies are evaluated in a *train/test framework* [65] following the methodological recommendations in [117]. learned policies are periodically evaluated during the learning process. Evaluation consists on running a number of test episodes under the greedy policy defined by the current estimation of the Q-matrix. During these test episodes, no learning is performed. In the case of D-RR-QL, the joint-policy being evaluated was composed using the procedure described in Subsection 6.4. For the training episodes, the action selection policy is $\epsilon-greedy$ in all the cases, and the same seed was used with all the algorithms to generate the random sequences of exploratory actions. In the following, we will call *valid greedy policies* as the test greedy policies able to reach the goal . The performance measures are:

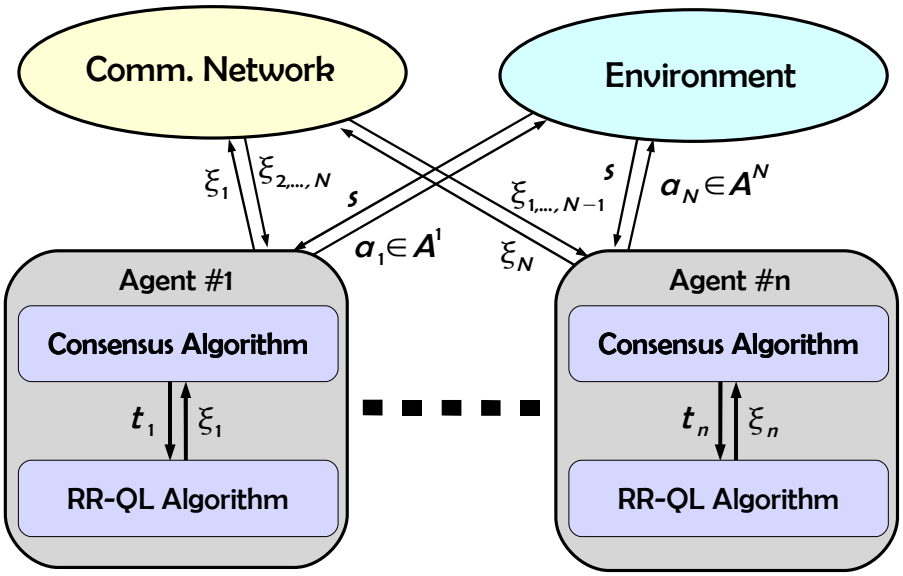


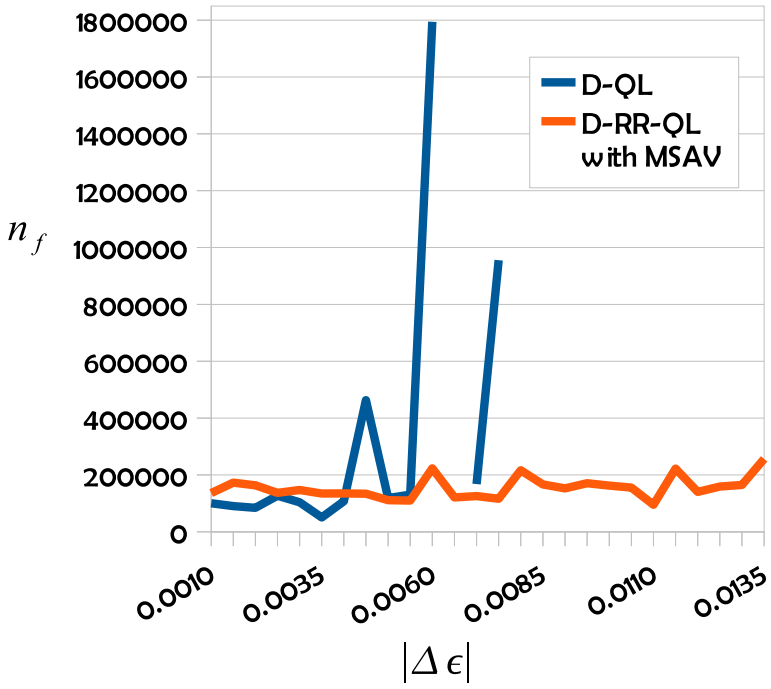
Figure 6.3: Control scheme of the Consensus implementation of the RR-QL Algorithm.

- n_g : #steps adding all episodes needed to reach the goal for the first time in a test episode,
- n_f : #steps adding all episodes needed to produce the first valid greedy policy,
- s_f : #steps taken by the first instance of the greedy policy test to reach the goal, and
- s_b : #steps taken by the best valid greedy policy to reach the goal.

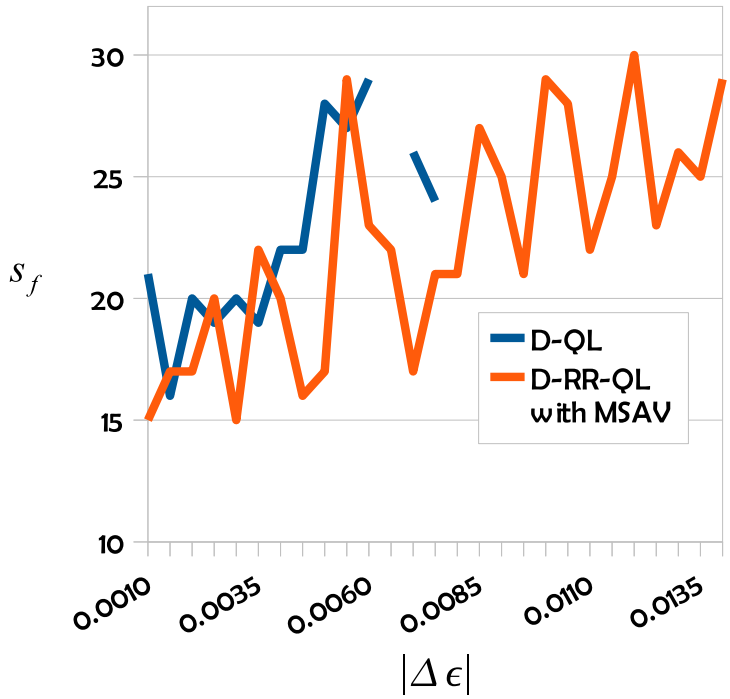
6.6.1 Multi-Agent Taxi Problem

The multi-agent taxi problem's definition does not include UTS, nevertheless it can benefit from the use of MSAV policies. State-action pairs leading to a negative reward are assumed not to be part of the optimal solution, therefore they can be safely discarded, allowing a much more efficient exploration of the state-action space. A single Veto-Module allows to learn the unsafe actions leading to the undesired state subspace, indicated by the four flags.

Initially we set $\epsilon = 1$, decreasing its value at fixed rate after each episode until it reaches 0. Aiming at a faithful comparison, we ran the full experiment for 26 different values of $\Delta\epsilon$ in the range $[-0.001, -0.0135]$. The higher the decrement magnitude, the faster the system goes from pure exploration to pure exploitation of the available estimations of Q-values.



(a) Plot of n_f.



(b) Plot of s_f.

Figure 6.4: Performance results of the D-QL and D-RR-QL on the multi-agent Taxi problem. Plot missing values correspond to inability to produce a valid greedy policy within allowed computational time.

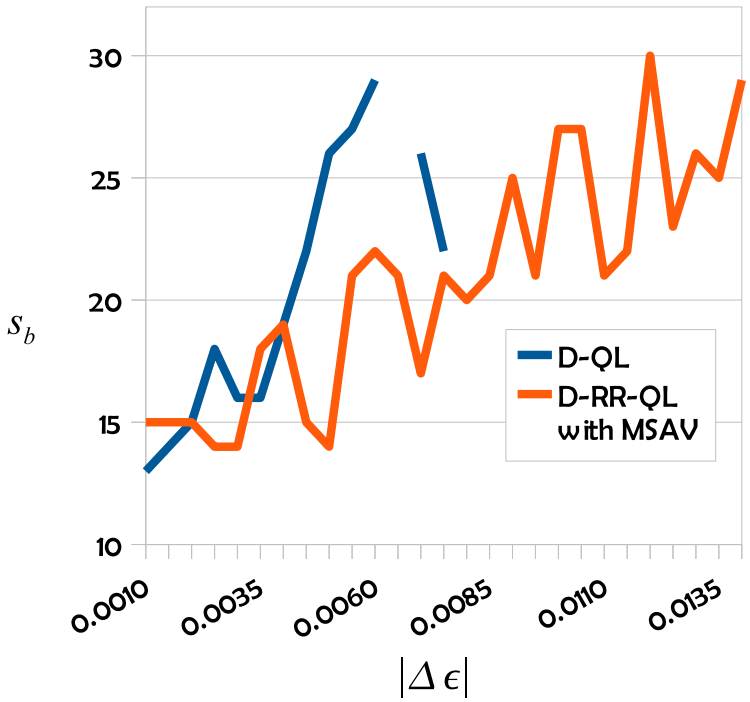


Figure 6.5: Performance results of the D-QL and D-RR-QL on the multi-agent Taxi problem. Plot missing values correspond to inability to produce a valid greedy policy within allowed computational time. Plot of s_b.

Figures 6.4a, 6.4b, and 6.5 provide plots of the three measured performance indicators: n_f, s_f and s_b. Because there are no abrupt termination conditions, n_g is of little significance and, thus, is not represented. The results show that D-RR-QL with MSAV was able to learn a valid greedy policy employing less exploration time than D-QL, which was not able to produce any valid greedy policy for values of $\triangle\epsilon < -0.0085$. Inspecting Figure 6.4a, the number of steps needed to provide the first valid greedy policy by D-RR-QL is almost invariant to the greediness parameter, while D-QL shows some peaks for lower values of $\Delta\epsilon$. Regarding the quality of the learned greedy policies, with restricted exploration time D-RR-QL with MSAV outperforms D-QL. As the exploration time increases, there is no significant difference.

6.6.2 Hose Transportation Problem

Next, we compared the performance of D-QL and D-RR-QL in the simplified hose transportation environment, which includes UTS. In the D-QL algorithm

agents only use relative positions because the remaining state variables are a linear function of them. On the other hand, D-RR-QL with MSAV allows the use of extended sets of state variables. Each of the four aforementioned negative rewards was fed to a specific module learning the veto of specific undesired state-actions using the value of the flag variables[39].

6.6.2.1 Exploration robustness

The first experiment aims to highlight the robustness introduced avoiding state-action pairs leading to UTS by the veto mechanism. The systems was set ($\epsilon = 1$, $\triangle\epsilon = 0$) for random exploration of the environment during 10^6 steps[2] with no designated goal position. Figures 6.6a, 6.6b and 6.7 show the number of different positions visited by the tip of the hose applying D-QL and D-RR-QL with MSAV on three system configurations: system with 2 robots, 3 robots, and 4 robots. The higher number of visited positions indicates that the system does not get stuck in UTS so more of the operational time is really devoted to the exploration of the environment.

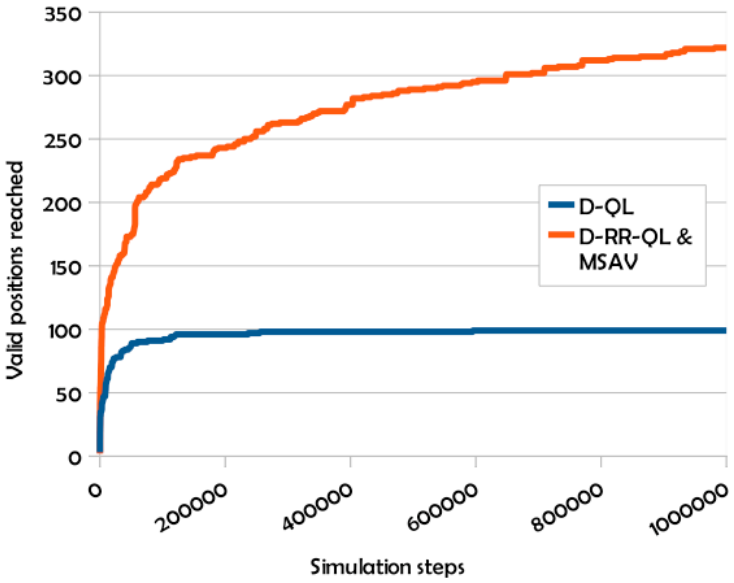
At the early state of the exploration process, both algorithms show a similar performance reaching quickly positions which can be reached from the starting position executing a small number of actions. Increasing exploration time, D-QL has a steady probability of falling in UTS impeding the on going exploration, so that the number of different visited positions saturates to a low value. On the other hand, D-RR-QL with MSAV avoids falling in UTS, so that the number of different visited positions grows steadily in time. The plots also show that exploring the state-action space safely is more difficult as the number of robots increases. Higher exploration robustness means a shorter time to the first visit to the goal position. Until this first visit, the agents do not perceive any positive reward, therefore early first visits are key for learning speedup.

6.6.2.2 Comparison of learned greedy policies

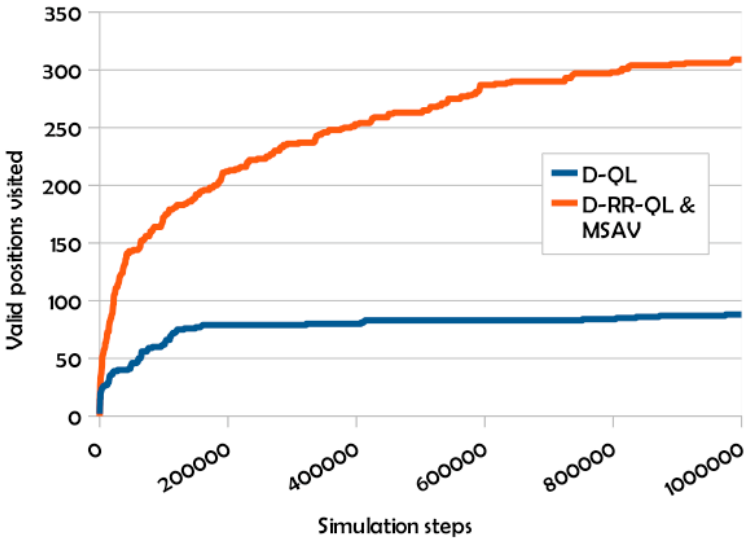
We select ten random system configurations, selecting a goal position and a number of robots $N \in \{2, 3, 4\}$ applying the learning algorithms to learn the state-action matrices. Parameter ϵ was initially set to 1 and it was decreased ($\Delta\epsilon = -0.001$) each time the goal position was reached. Each 100 episodes, the policy learned was tested applying the greedy policy given by the estimated Q-values . We set a computational time limit of 10^7 steps adding all episodes.

Table 6.3 shows the measured performance of both D-QL and D-RR-QL with MSAV. The "n/a" entries mean that no result was obtained after the computational time limit expires. From the ten experimental system configurations,

[2] A step is a individual action in the case of D-RR-QL, and the simultaneous execution of the joint-action in D-QL)



(a) System with 2 robots.



(b) System with 3 robots.

Figure 6.6: Robustness of the exploration of D-QL and D-RR-QL with MSAV: number of safe positions visited by the tip of the hose.

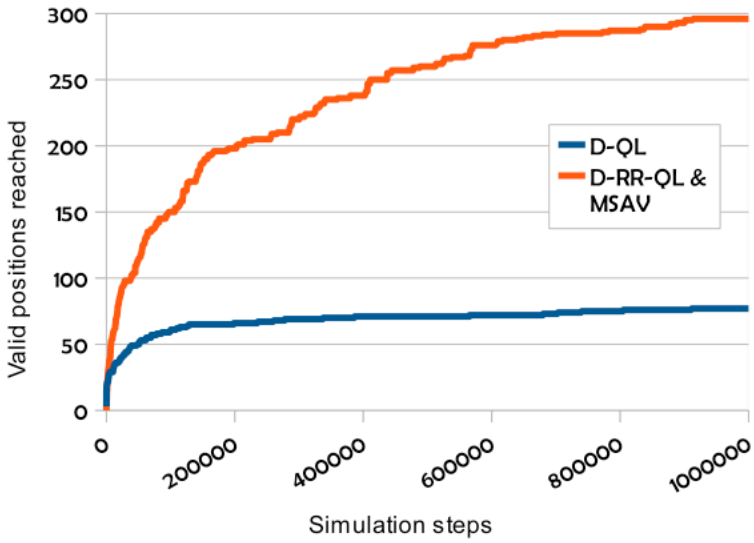


Figure 6.7: Robustness of the exploration of D-QL and D-RR-QL with MSAV: number of safe positions visited by the tip of the hose with 4 robots.

	D-QL				D-RR-QL			
conf.	n_g	n_f	s_f	s_b	n_g	n_f	s_f	s_b
1	131,161	1,556,186	7	7	25,479	636,924	8	7
2	4,954,452	n/a	n/a	n/a	128,756	4,993,018	13	13
3	6,099	n/a	n/a	n/a	183,450	6,104343	14	9
4	9,666	257,562	10	10	9,272	159,358	11	10
5	19,679	372,434	10	8	2,996	235,496	11	8
6	1,067,789	n/a	n/a	n/a	144,723	9,422,379	14	11
7	n/a	n/a	n/a	n/a	732,088	n/a	n/a	n/a
8	n/a	n/a	n/a	n/a	18,912	1,200,283	16	14
9	2,083	4,803	5	5	2,902	20,028	5	5
10	n/a	n/a	n/a	n/a	479,644	n/a	n/a	n/a

Table 6.3: Results of learning from ten different initial configurations.

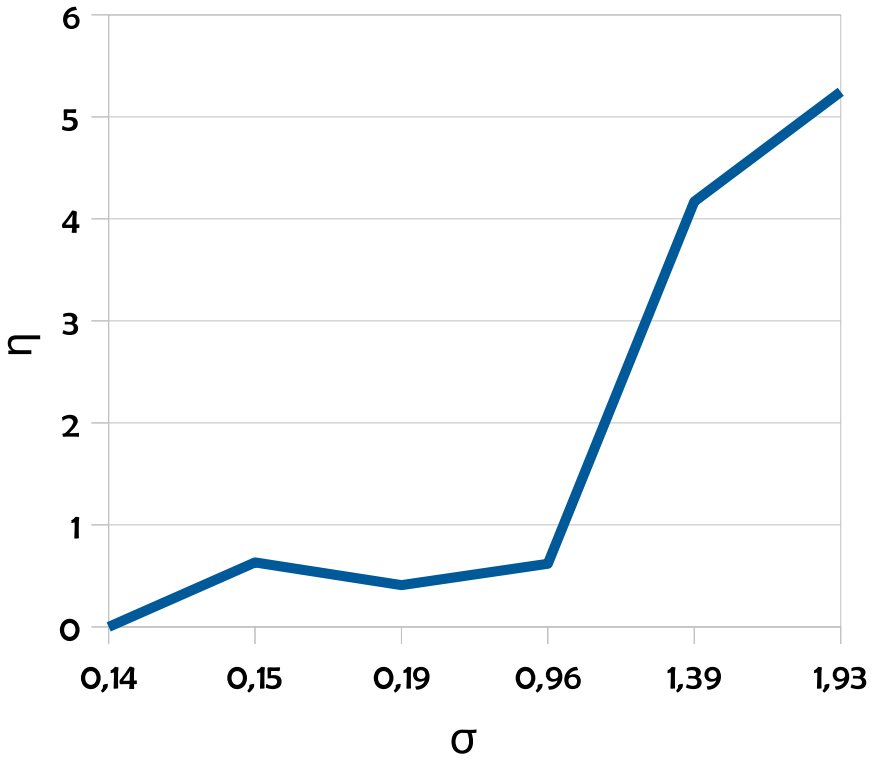


Figure 6.8: Learned greedy policies: relation between time-differences to discover a goal (σ) and time-differences to learn a valid greedy policy (η).

D-QL was not able to learn a valid greedy policy in 60% of the experiments. For 30% of the experiments , the goal position was not reached after 10^7 steps. On the other hand, the D-RR-QL with MSAV reached the goal in 100% of the experiments and was able to learn a valid greedy policy in 80% of the experiments.

Regarding the quality of the first valid greedy policy found by the algorithms, measured by the number of steps needed to reach the goal position per episode, D-QL policies are slightly better than D-RR-QL ones. The relative difference was 9.58% (standard deviation = 6.86%), however D-RR-QL was 1.47 times faster (standard deviation = 0.91) than D-QL. Note that this only refers to those four configurations in which D-QL was able to learn a valid greedy policy. It is also interesting to note that s_b was the same for both algorithms, meaning that the composition of joint-actions for D-RR-QL to obtain the greedy policies in the evaluation periods was accurate, reassessing the sensibility of the assumptions made in Subsection 6.4. To compare the convergence times of D-QL and D-RR-QL we have defined the following ratios:$\sigma = \frac{n_g^{D-RR-QL}}{n_g^{D-QL}}$ and $\eta = \frac{n_f^{D-RR-QL}}{n_f^{D-QL}}$. Values of σ below 1 mean that D-RR-QL required less steps than D-QL to reach the goal position for the first time. Values of η below 1 mean that D-RR-QL required less steps than D-QL to produce a valid greedy policy in test episodes. It can be seen in Figure 6.8 that most of the times D-RR-QL with MSAV outperforms D-QL.

6.7 Conclusions

In this chapter, we have presented a novel distributed approximation of the single-agent Q-Learning algorithm. This approach inherits some of the advantages of Distributed Q-Learning over usual MARL approaches, but unlike Distributed Q-Learning, it is able to take advantage from Modular State-Action Veto policies and thus, expected to perform better in over-constrained environments. First, we have presented a theoretical framework (C-RR-SG), in which agents select and take actions following some predefined order. A communication-free distributed implementation of single-agent (D-RR-QL) has been proposed and proof of convergence to the optimal policy on a C-RR-SG has been given. We have also presented a message-based procedure to approximate the optimal policy of the original SG the learned C-RR-SG is derived from. To that end, local Q-values learned with D-RR-QL are used. A consensus-based implementation has been proposed to deal with turns and, finally, results from computational experiments have been presented.

Computational experiments show that D-RR-QL can provide a valid joint-policy approximating the optimal policy faster than D-QL in over-constrained systems using MSAV. Furthermore, the theoretical background holding D-RR-QL is not limited to deterministic MDPs and can also cope with stochastic ones. Communication requirements to learn local Q-values ($O(1)$ messages each step if communications are used, 0 messages otherwise) and to agree on

a joint-action (each step requires $O(N)$ messages) are minimal if compared to coordinated multi-agent approaches such as Coordinated-QL. Although anytime mechanisms can give an acceptable response faster than exhaustive search, they are likely to give suboptimal results. Besides, determining coordination requirements introduces added computational requirements. As a potential line of future work, we will study the error introduced by our approach with respect to the original SG and empirically study its applicability to stochastic environments. We also plan to investigate more general ways to learn avoiding reaching undesirable states.

It is our belief that distributed implementations of single-agent RL algorithms constitute a very interesting alternative to general MARL because they overcome some of the inherent issues related to multi-agent learning. Our experiments have shown that composing joint-policies using local state-action values is a promising venue of work in the area of distributed RL.

Chapter 7

Partially Constrained Models

Transfer learning is a hierarchical task refinement approach to the RL of complex tasks. Tasks are learned in increasing order of complexity, the results of learning in the simpler tasks (source task) are used as starting point for learning in the more complex tasks (target task). In this chapter we deal with a hierarchy of constrained systems, where the source task is an under-constrained system, hence called Partially Constrained Model (PCM). We propose a theoretical background for the hierarchy of training refinements, showing that the effective action repertoires learnt on the PCM are maximal, and that the PCM-optimal policy gives maximal state value functions. We apply the approach to learn the control of the hose transportation by a team of robots, the paradigm of Linked Multicomponent Robotic Systems (L-MCRS). The target task corresponds to the control of an accurate GEDS hose model, described in Appendix A. The PCM is obtained simplifying the hose model. Learning results of the PCM Transfer Learning show an spectacular improvement over conventional Q-learning on the target task.

The chapter is structured as follows. First, in Section 7.1 we give an introduction to the Transfer Learning techniques. Section 7.2 introduces the PCM Transfer Learning, proving some theoretical results. Section 7.3 reports experimental results on the hose transportation task based on the GEDS model. Finally, Section 7.4 offers some conclusions.

7.1 Introduction

Transfer Learning (TL) is the process of transferring the knowledge gained on a specific task to another related task. Nearly all the RL approaches to learn the control of robotic systems usually imply one of the most basic forms of TL. Because robotic units can wear down from the repeated use that RL algorithms require to learn the task, most researchers use first a computational simulation model on which the agent learns the basic skills, and then the agent completes the learning process on the real world task.

Tasks are modeled by Markov Decision Processes (MDP). For MDP with many undesired terminal states (UTS), a convenient simplification consists in removing state variables related with physical constraints, obtaining a *Partially Constrained Model* (PCM) from the target MDP to be the source MDP[1]. The PCM contains much less UTS allowing faster learning of optimal policies. The PCM Transfer Learning (PCM-TL) performs RL on the PCM for the initial training steps. The knowledge acquired is transferred to speed up learning on the more complex target. The source and target may have different underlying dynamic models (state transition functions) meeting certain requirements. The target may have additional termination states not included in the PCM, and therefore, different reward functions. If the source task is a PCM of the target, the initial exploration phase of the RL in the target environment is more effective in PCM-TL than in the direct application of RL. Thus, the PCM-TL system is expected to learn the optimal policy faster than the direct RL method. Features of PCM-TL distinctive from other TL approaches are the following:

- The source task is a PCM of the target task. It is human designed, we do not know any automated procedure to generate feasible PCMs of a given task.
- Source and target tasks may have different state spaces, reward and transition functions.
- A state variable mapping is provided by the theoretical framework
- Knowledge transference consists of the Q-values and action repertoire sets $A^e(s)$ learnt on the source task
- The learning mechanism are TD algorithms.
- Learning performance is measured as the rate of goal states reached at least once for a set of initial configurations.

[1] We will obviate "MDP" when it s clear from the context.

7.2 Partially Constrained Model Transfer Learning

This section defines a PCM of a target, giving conditions for its correct construction. Then we define the mapping of the results of learning on the PCM to the starting point of the target MDP learning. We give also two theoretical results on the improvement of convergence obtained by PCM-TL.

Definition 11. Given a source MDP $< S_s, A, P_s, R_s >$ and a target MDP $< S_t, A, P_t, R_t >$, we say that the source is a PCM of the target if the following properties are met by the source:

1. Property 1: $S_t = S_s \times S_Y$, where S_Y is the state subspace spanned by a subset Y of target state variables removed to obtain the source state variables. Therefore, the source's state variables are a proper subset of the set of state variables of the target.

2. Property 2: Preservation of the transition probability mass. The transition probabilities from a state in the target MDP marginalized relative to the state variables in Y are the same as in the corresponding reduced state in the source. Formally:

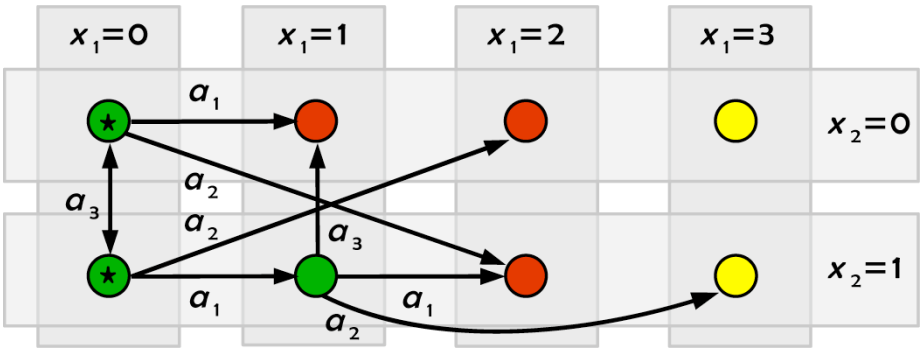
$$\forall s, s' \in S_t;\ \forall a \in A;\ \sum_{[t]_{S_s}=[s']_{S_s}} P_t\left(s, a, t\right) = P_s\left(\left[s\right]_{S_s}, a, \left[s'\right]_{S_s}\right),$$

   where $[t]_{S_s}$ denotes the projection of target state t into the subspace spanned by the state variables of the source.

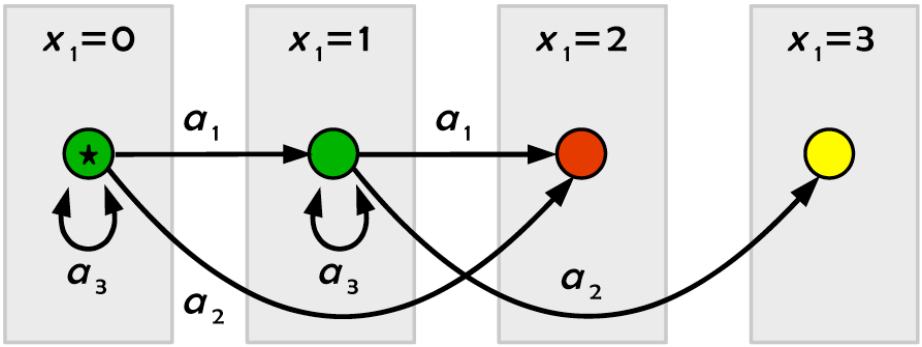
3. Property 3: $\forall s \in S;\ R_t\left(s\right) \geq 0 \Rightarrow R_t\left(s\right) = R_s\left(\left[s\right]_{S_s}\right)$. Goal and transition states in the target correspond to goal and transition states in the source MDP with the same reward function.

4. Property 4: $\forall s \in S; R_t\left(s\right) < 0 \Rightarrow \left(\left[R_t\left(s\right) = R_s\left(\left[s\right]_{S_s}\right)\right] \vee \left[R_s\left(\left[s\right]_{S_s}\right) = 0\right]\right)$. UTS in the target correspond to either transition or undesired terminal states in the source.

We illustrate the mapping of a target into a PCM with two examples of deterministic and stochastic target MDPs in Figures 7.1 and 7.2, respectively. Colored circles correspond to MDP states, arrows to feasible transitions, the transition probabilities are given in brackets. Starred circles are initial states, green circles are transition states, red circles are undesired termination states, and yellow circles are goal states. Figure 7.1a represents the deterministic state transitions for each of the actions $a \in \{a_1, a_2, a_3\}$ and Figure 7.1b represents an example of PCM derived from it. All required properties in Definition 11 hold: (1) the PCM state variables ($X_s = \{x_1\}$) is a subset of the target's state variables ($X_t = \{x_1, x_2\}$), (2) transitions in the PCM and those projected to the target's state subspace are equal, (3) all goal and transition states in the

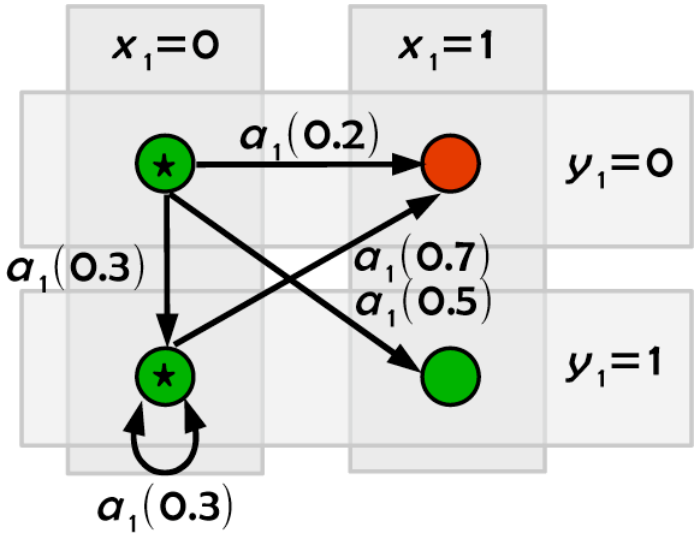


(a) A deterministic target MDP.

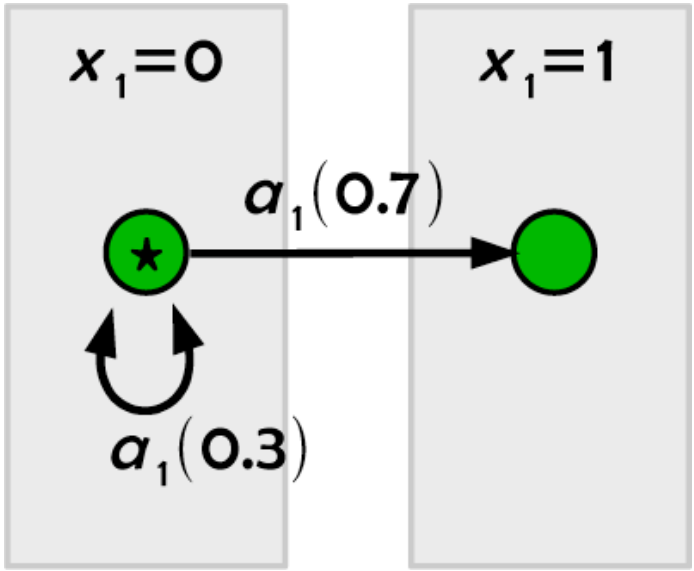


(b) PCM derived from the target MDP removing state variable x_2.

Figure 7.1: An example of a PCM from a deterministic target MDP.



(a) A stochastic target MDP.



(b) PCM derived from the target MDP removing state variable y.

Figure 7.2: An example of a PCM from a stochastic MDP.

target task have the same category in the PCM, and (4) undesired states in the target task include those in the PCM.

Figures 7.2a and 7.2b depict an stochastic target MDP and the PCM obtained removing state variable y, respectively. All properties of Definition 11 hold. Specifically, Property 2 is also fulfilled: state transitions in the projected subspace of the PCM are the sum of the probabilities of the transitions in the original target.

7.2.1 Task mapping

Task mapping function specifies how to make transference of learning results from the source to the target. From the previous definition, in general $S_t \neq S_s$. Property 1 ensures that S_s is a subspace of the target state space S_t. Therefore, we initialize the Q-table of the target task $Q_t(s,a)$ with the Q-values learnt from the source task $Q_s(s,a)$:

$$Q_t(s,a) = Q_s\left([s]_{S_s}, a\right), \tag{7.1}$$

regardless the value of state variables not present in the source. The action repertoires are likewise mapped:

$$A_t^e(s) = A_s^e\left([s]_{S_s}\right), \tag{7.2}$$

where A_s^e and A_t^e, respectively, represent the action repertoires learnt on the source and the target.

7.2.2 Theoretical results

We prove in this section two theoretical results about PCM-TL. First, that the effective action repertoire does not grow going from the source to the target, meaning that new vetoes can be discovered, but no vetoed action in the PCM can become a safe action in the target, thus the learning process performed on the PCM does not need to be revised. Second, the greedy action selection on the optimal Q-values learnt from the PCM provides an upper bound for the state value functions. Therefore, learning in the target task does not increase state values, providing only refinements due to state space increased resolution.

Proposition 4. *Let $< S_s, A, P_s, R_s >$ be a PCM of a target MDP $< S_t, A, P_t, R_t >$. For all states $s \in S_t$, the effective action repertoires in the target will be a subset of the effective action repertoires in the projected state in the PCM:*

$$A_t^e(s) \subseteq A_s^e\left([s]_{S_s}\right).$$

Proof. Let us consider a state-action pair (s,a), $s \in S_t$, $a \in A_t^e(s)$ in the target, leading to a state s' with $P_t(s,a,s') > 0$ with positive reward $R_t(s') \geq 0$. Property 2 of PCM implies that $P_s\left([s]_{S_s}, a, [t]_{S_s}\right) = 0$ for any state $t \in S_t, t \neq s'$ such that $P_t(s,a,t) = 0$, meaning that only projections of reachable states in the target task will be reachable in the source task from state s. On the other

hand, property 3 of PCMs guarantees that all reachable states s' with $R_t(s') \geq 0$ have corresponding $R_s\left([s']_{S_s}\right) \geq 0$. Therefore, all transitions in the PCM such that $P_s\left([s]_{S_s}, a, [s']_{S_s}\right) \geq 0$ have positive reward $R_s\left([s']_{S_s}\right) > 0$, and, thus, $a \in A_s^e(s)$. □

The task mapping procedure described in Equation (7.2) ensures that all unsafe actions vetoed in the source task will also be vetoed from the start in the target task. Thus, the agent needs only to execute once the actions in $A_s^e(s) - A_t^e(s)$ to learn the correct effective action repertoire $A_t^e(s)$. This reduction in the size of sets of effective action repertoires to be learnt implies a large computational requirement reduction in many problems. Proposition 4 implies also that a vetoed action in the source task will never revert to be a safe action in the target task.

Proposition 5. *Let* $< S_s, A, P_s, R_s >$ *be a PCM of a target MDP* $< S_t, A, P_t, R_t >$. *PCM optimal* $Q_s^*(s,a)$ *values and* $A_s^e(s)$ *sets are learnt. After transferring them to the target task applying Equations* (7.1) *and* (7.2)*, greedy action selection in* $A_t^e(s)$ *according to the source task optimal Q-values, i.e.* $\pi_t^g(s) = \arg\max_{a \in A_t^e(s)} Q_s^*\left([s]_{S_s}, a\right)$ *in the target gives an upper bound for the optimal state values in the target , i.e.* $V_t^{\pi_t^g}(s) \geq V_t^*(s)$.

Proof. Greedy selection on the optimal Q-values of the source task $\pi_s^g(s) = \arg\max_a Q_s^*(s,a)$, $s \in S_s$, is the optimal policy for the source task, i.e. $V_s^{\pi_s^g}(s) = \max_\pi V_s^\pi(s)$. Greedy action selection in the target task $\pi_t^g(s)$, $s \in S_t$, after transference of the optimal Q-values is equivalent to the source optimal policy in the state projections, i.e. $\pi_t^g(s) = \arg\max_a Q_s^*\left([s]_{S_s}, a\right)$. Properties 3 and 4 of the definition of PCM imply that $R_t(s) \leq R_s\left([s]_{S_s}\right)$. This result, together with Property 2 implies that, after applying RL in the target task obtaining an optimal policy $Q_t^*(s,a)$, we will find $Q_t^*(s,a) \leq Q_s^*\left([s]_{S_s}, a\right)$ for all state-action pairs in the target. Notice that $V^*(s) = \max_a \{Q^*(s,a)\}$, therefore $V_t^{\pi_t^g}(s) \geq V_t^*(s)$. □

After the transference of optimal Q-values from the source task to the target task, greedy selection of actions based on the transferred Q-values gives the upper bound for the value function of states having a non-null projection in the PCM state space. However, the state space expansion introduces new transitory, goal and undesired terminal states, some of them without correspondence with the PCM states. Learning in the target task aims to assign value to these new states, refining the value given to the PCM states in the new context. Proposition 5 implies that Q-values obtained by RL in the source task are maximal, so that RL in the target task will preserve them unless affected by the appearance of undesired terminal states, i.e. that an undesired terminal state is projected into a transition PCM state.

7.3 Computational Experiments

We have applied PCM-TL to the hose transportation problem using the GEDS model with two, three and four robots ($N = 2, 3, 4$), each of which implemented an agent. Shared reward signals were used to enforce cooperation.

Target MDP The system dynamics are accurately simulated by the GEDS model, but it is computationally very expensive[2]. In the target task, all the state variables defined in Appendix B were used. Those state variables used in the simplified hose transportation task which are not defined in the GEDS model were calculated on-line (with the same mapping function used in the PCM) for the integer coordinates of each robot on the grid, $c(P_i)$.

Source MDP: PCM We use *a priori* knowledge on forbidden behaviors of the robots while maneuvering the hose, to build a PCM of the system. The PCM removes the spline representation, modeling the hose as a set of line segments. The PCM reward signals are designed taking onto account the physical constraints as follows: (a) hose break is modeled by any one of the line segments becoming bigger than the nominal length of a segment of the target hose, (b) a robot gets out of the grid when the next action drives it outside the space, (c) a robot steps over a line segment representing the hose when it reaches an spatial cell crossed by a line segment, and (d) robots collide when they reach the same spatial cell. The PCM allows to compute about $10,000$ simulation steps per second. Although hundreds of episodes need to be conducted before a goal state can be reached, this takes in all cases less time than simulating a single step in the target task.

The agents were trained using D-RR-QL with MSAV. Four different modules were used with different sets of state variables (Appendix B):

- *Goal* module: the state variables observed by each instance of this module were r_i^n and r_i^d, $i = 1 \ldots N$. Note that, unlike the experiments in Chapter 4, all instances of the modules can observe the relative positions of all the robots. Whenever the tip of the hose reaches the goal, a positive reward is given to all instances of this module in all the agents, otherwise a neutral value.

- *Veto-Distances* module: all instances of this module modeled the state as the combination of values of r_i^n and r_i^p. They receive a negative reward every time any of the two hose segments exceeds the maximum allowed length, and a neutral one otherwise

- *Veto-Collisions* module: this module learns to avoid collisions between robots and hose segments modeling the state as the combination of the four boolean flags $o_i^{up}, o_i^{down}, o_i^{left}$ and o_i^{right}, each of which indicates whether

[2]The Matlab implementation on an off-the-shelf desktop computer takes about 39 seconds to simulate a single step for the simplest configuration ($N = 2$).

there is an obstacle (robot or hose segment) within one time-step reach in a direction corresponding to one of the allowed actions. Whenever a collision occurs a negative reward is received, otherwise a neutral one.

- *Veto-InGrid* module: they model the state as the absolute position of the robot controlled by the agent, p_i. A negative reward is received if the position is outside the allowed bounds, neutral otherwise.

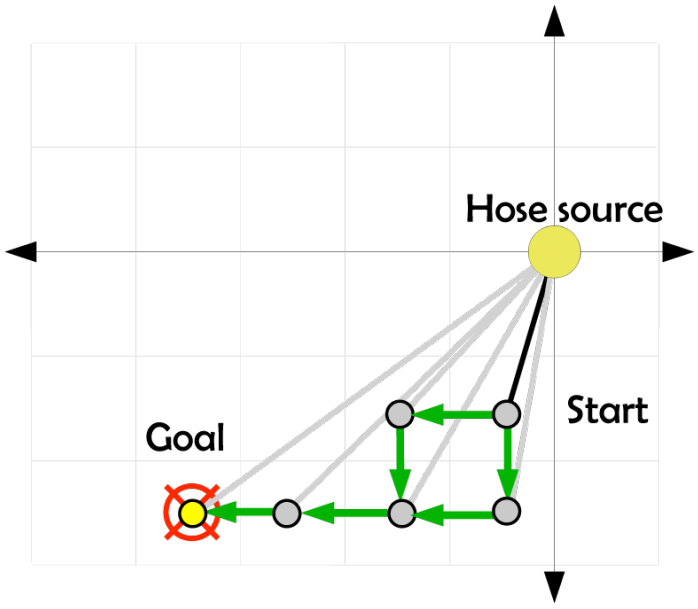
The state variables $o_i^{up}, o_i^{down}, o_i^{left}, o_i^{right}$ were generated using the line segment model of the hose. Spline-related flags $f_{i,j}$ are purposefully omitted.

Figure 7.3 shows an instance of the configuration of (a) the PCM and (b) the GEDS-based target task in a one robot case. Green arrows indicate safe robot movements (actions). The red arrows indicate unsafe movements, actions leading to an undesired termination state. In the PCM configuration, the robot can initially move left and down because the hose modeled as a line segment does not interfere with the advance. In the target task, the GEDS model has the hose invading one of the possible robot motion cells, therefore the robot cannot move left because it would step on the hose.

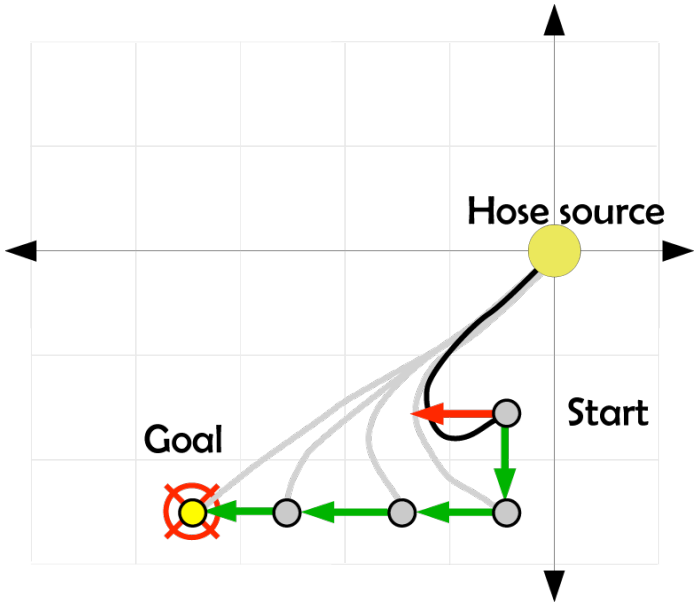
Experimental design For each of the three different values of N, 30 different initial configurations were randomly generated. For every configuration, the D-RR-QL learning system was first run on the PCM before the results of learning were transferred to the target MDP following Equations 7.1 and 7.2. The time allowed for learning on the PCM was increased linearly with the number of robots: 30, 60 and 90 seconds for the 2-robot, 3-robot and 4-robot systems, respectively. After transference of the PCM learning results, three episodes of Q-learning with a greedy policy ($\epsilon = 0$) were realized for each initial configuration in the target task, using the GEDS simulation to obtain the system's reward. For comparison, Q-learning with a pure exploratory policy (ϵ-greedy with $\epsilon = 1$) was executed on the target task for the same amount of processing time as the PCM-TL complete cycle.

7.3.1 Results

Figures 7.4a, 7.4b and 7.5 show the percentage of episodes ending up in goal states (the tip of the hose reaching the goal) during Q-learning on the target task (blue curve) and the exploratory Q-learning without PCM-TL (red curve) for systems with 2, 3 and 4 robots, respectively. Time spent learning in the PCM is added to the time axis of the PCM-TL results for fair comparison with the non PCM-TL approach. The problem gets harder as the number of robots increases because of the increasing number of constraints. However, there is a paradoxical result, the PCM-TL reaches better results for the case with 4 robots than in the case with 3 robots. In any case, learning in the target task after PCM-TL improves steadily, while the non-PCM-TL remains stuck. The exploratory Q-learning without PCM-TL was only able to find the goal in the *easiest* configurations, that is, in the cases were the goal could be reached taking

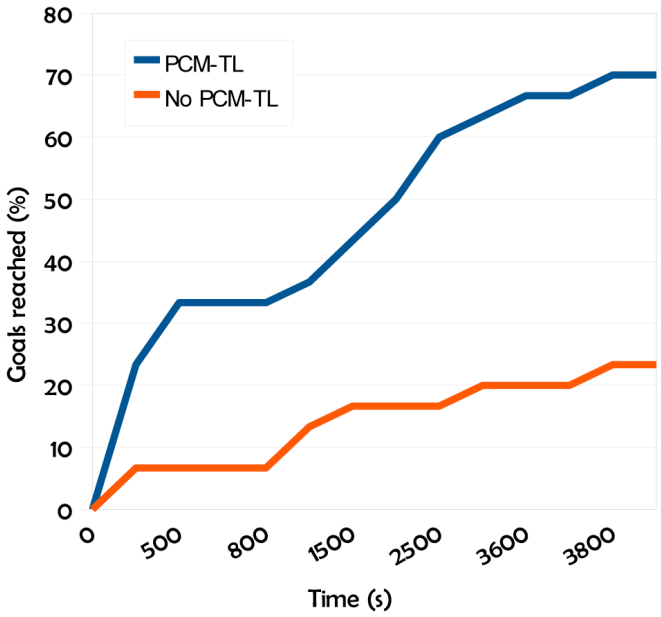


(a) Simplified PCM.

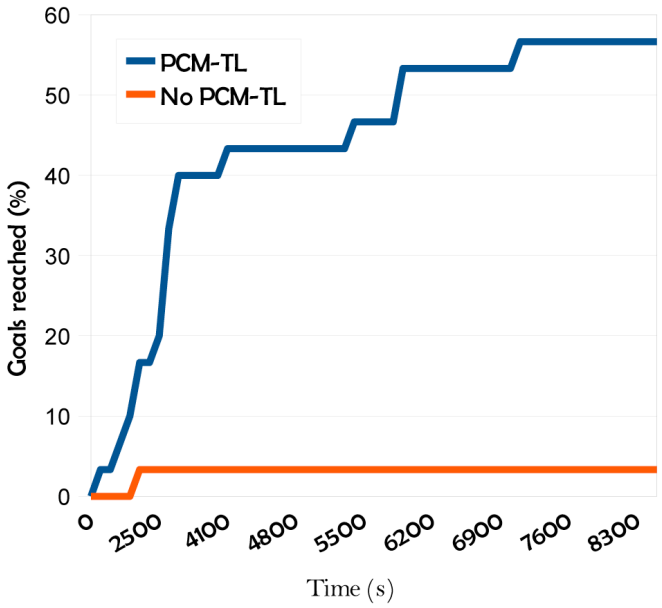


(b) GEDS simulation environment.

Figure 7.3: An example of the different constraints obtaining in the hose transportation problem depending on the simulation model used.



(a) 2 robots



(b) 3 robots

Figure 7.4: Experimental results. Percent of goal states reached with and without PCM-TL for different configurations.

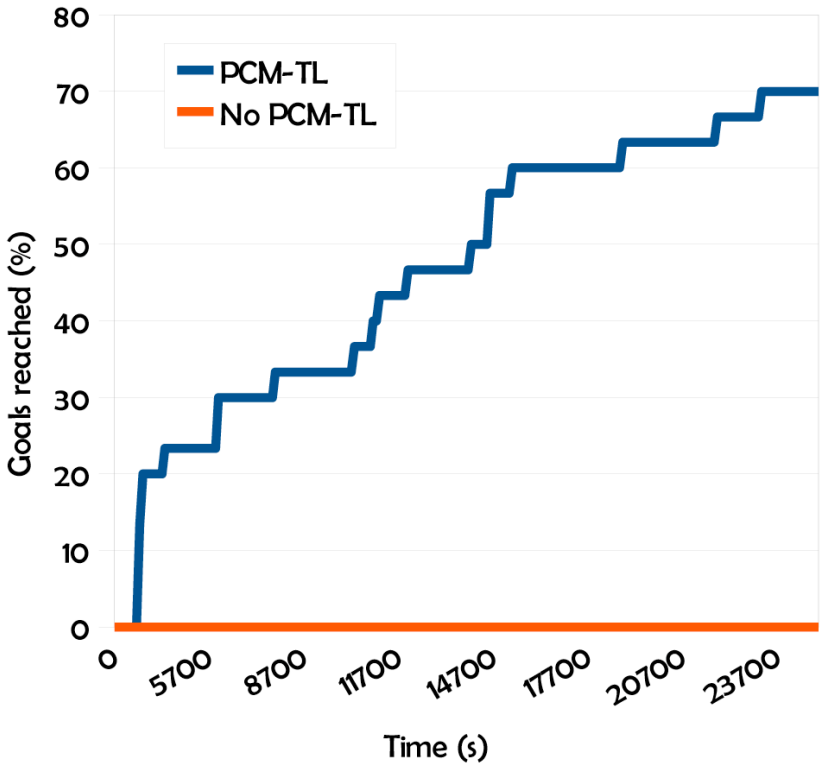


Figure 7.5: Experimental results. Percent of goal states reached with and without PCM-TL for 4 robots.

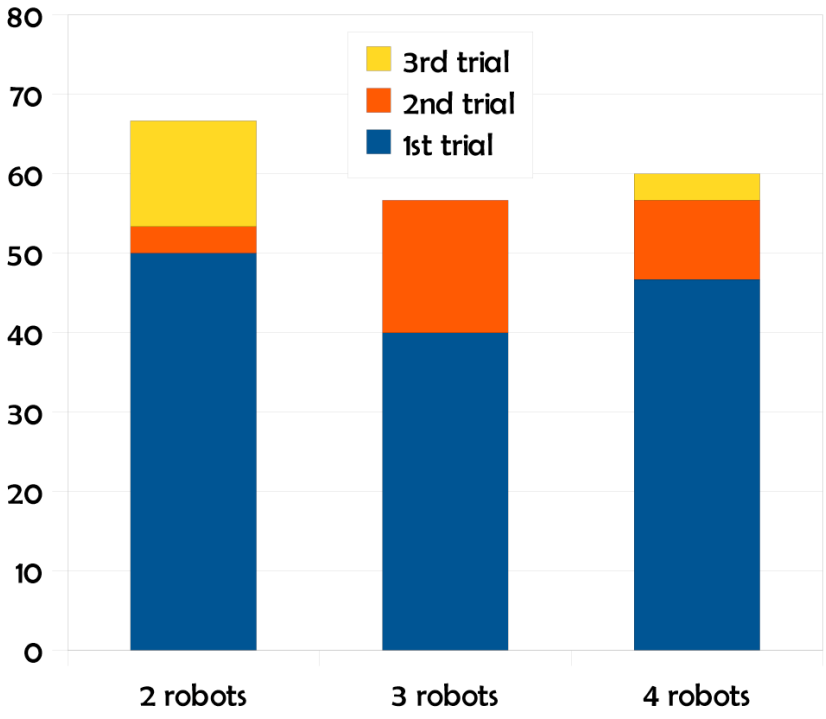


Figure 7.6: Trials needed to reach the goals.

a small amount of steps from the initial position: 7 goal states were reached in the allowed time for $N = 2$, only one was reached for $N = 3$, and none for $N = 4$. This is in a dramatic contrast with the 70%, 60% and 70% of goal states reached by the PCM-TL after the same amount of time on the same systems.

Figure 7.6 represents the percentage of goals reached on the 1^{st}, 2^{nd} and 3^{rd} episode of Q-learning with greedy policy tried from each initial configuration. For all the different systems with 2, 3 and 4 robots, most of the goal states were reached in the 1^{st} run. As a consequence of Proposition 5, when the optimal policies learnt on the PCM lead to episode trajectories which do not encounter unknown undesirable states in U_t, the first episode trial reaches the goal state following the PCM-optimal policy in the target task. Half of the configurations (50%) were solved optimally for $N = 2$, 40% for $N = 3$ and 46.67% for $N = 4$. The remaining configurations successfully solved reaching goal states were the result of the Q-learning episode trials refining the estimation of Q-values according to the new undesired terminal states.

A video showing the comparative learning evolution in the real time reference frame can be found in `http://www.ehu.es/ccwintco/index.php/Borja-videos#Partially_Constrained_Models_.282011.29` of both the PCM-TL and the conventional Q-learning trained directly on the GEDS simulation for the 3 robots case from the same initial configuration. The image sequence on the left hand of the video shows first the learning on the PCM, second, the policy learned on the PCM is used as starting point for learning on the accurate GEDS simulation. The right hand image sequence shows the evolution of the Q-learning on the GEDS simulation. The conventional Q-learning does not reach the target position in the time employed by the PCM-TL to generate an optimal policy.

7.4 Conclusions

Reinforcement Learning in systems with many physical constraints is done on MDPs with many UTS involving much unfruitful computation. To speed up learning, this chapter presents a new Transfer Learning method based on the definition of Partially Constrained Models (PCM) on state-action veto policies, which is likely to reduce the exploration time required to reach a goal. We have given the properties that a reduced MDP must meet to be a PCM of the target MDP. We have proved that the results of Q-learning on the PCM are optimal when transferred to the target MDP in two ways: (a) the effective action repertoires are maximal, and (b) the optimal policy in the PCM gives an upper bound of the state value function in the target task.

The application domain is the control of a multi-robot system performing hose transportation. Experiments have been conducted with a computationally expensive simulator (GEDS) with results showing a huge improvement on the learning performance with the same allowed computational time of the PCM-TL over a conventional Q-learning started with random exploration of the state-

action space.

Future works will address the systematic definition of PCM from given target tasks, and the analysis of the relation between the optimal policy inherited from the PCM and the ones obtained after Q-learning in the target task starting from it, in order to ascertain the improvement that can be expected.

Chapter 8

Undesired state-action prediction

This chapter introduces Undesired state-action prediction (USAP) based on supervised learning techniques to obtain a hybrid classifier-aided reinforcement learning scheme. A modified version of Q-Learning learns the Q-values of the safe state-action pairs, which are selected by a classifier trained to predict UTSs.

Section 8.1 gives an introduction and some necessary background on Supervised Learning. Section 8.2 presents a thorough description of the USAP algorithm and an extension to multi-agent framework reinforcement learning using D-RR-QL. Section 8.3 reports our computational results in two different domains: the GEDS hose transportation task and the multi-agent taxi domain. Finally, Section 8.4 offers our conclusions.

8.1 Introduction

Supervised Learning (SL) algorithms are provided a *training set* of data, which are instances of labeled samples $< x_{i,1}, \ldots, x_{i,k}, l_i >$, $i = 1, \ldots, n_t$, where l_i is the correct output for sample $(x_{i,1}, \ldots, x_{i,k})$. Each of the n_t training samples has k attributes. The goal of the learning algorithm is to infer the functional relation between the input and output data. The generalization of the learned function is its accuracy predicting the correct output for a yet unseen input. The output could be a discrete valued variable (then we have a classification problem) or a continuous valued variable (then we have a regression problem).

In order to provide estimations of the generalization power of the function learned, first, the algorithm is presented the training data, the input-output function is estimated and then, the test data is processed to obtain predictions of its output values. The *test set* and the *training set* are independent data-sets to guarantee the statistical properties of the generalization prediction. Different performance statistics are obtained comparing the classifier's output with the supervised labels.

For an algorithm to be applicable in on-line classification applications, it is desirable that the algorithm is able to update its classification function with little computation when new instances are presented to it. *Incremental algorithms* are those able to take advantage of the previously model to generate the next one, without the need to recompute it again from scratch.

Many SL classification algorithms have trouble working on *unbalanced* databases, where some classes have more instances than others. If too few samples of a given class are available, the classifier might not be able to find appropriate boundaries for the classes due to the bias introduced toward the most frequent class. Another issue that must be taken care of is the domain of definition of the variables, and/or its scale. Some algorithms work best with *normalized* attribute values (in the interval $[0, 1]$). Their performance may degrade dramatically when the input variables have widely different value scales.

In this chapter, we will consider 2-class classifiers, where instances will be of class c_0 or c_1. The class c_0 corresponds to *safe* state-action pairs, and c_1 is the label of those pairs which are expected to probably lead to an undesired transition. Under the RL framework, undesired state transitions are described by negative rewards, and labels can be mapped using the mapping function $\omega : \mathbb{R} \longrightarrow \{c_0, c_1\}$, where

$$\omega(r) = \begin{cases} c_0 & \text{if } r \geq 0, \\ c_1 & \text{otherwise.} \end{cases}$$

Multi-classifiers Also known as classifier ensembles, a multi-classifier is built using a set of M different classifiers. The desired output is determined by a fusion method which receives the output of all the classifiers in the ensemble. Numerous examples of ensembles that outperform individual classifiers can be found in the literature [67, 71]. This approach is based on the assumption that the fusion of several weak (under-performing) classifiers can lead to an improved classifier. Weak classifiers play the role of experts focused on a sub-problem which perform badly on the global problem. If the fusion process is able to determine which expert is appropriate to a test data, then we obtain the desired accuracy improvement. Ensembles are commonly built using different perturbation methods to introduce diversity in the individual classifiers:

- Input perturbation. Each classifier is trained with a different training set. These can be different subsets of the original set or modified versions of the original set. Examples of ensembles built using input perturbation are bagging, arcing and boosting.

- Attribute perturbation. New training sets are built selecting different attribute subsets. Some examples are random subspace, cluster-based pattern discrimination and input decimated ensemble.

- Classifier perturbation. The same training set is used for all classifiers, but they have different parameters or belong to a different class of classifiers.

- Hybrid methods. More than one class of perturbation is used. Examples: random forests, rotation forests, rot-boost.

Random Forests One of the most popular multi-classifiers is Random Forest [18], consisting of a collection of tree-structured classifiers $h(x, \Theta_k)$, $k = 1, ..., n$ where Θ_k are independent and identically distributed random vectors. This ensemble is built using both input perturbation and attribute perturbation: each tree is built using bagging (its training set is randomly draw with replacement from the original set), and a random subset of attributes. The rationale of this random selection is to diminish the correlation between trees of the same forest. Using the CART methodology, trees are grown adding splitting nodes defined on a randomly selected small subset of the tree attributes. The class of an input instance is then predicted by each tree evaluating the attributes at each node. The forest's output is then selected by majority voting among all trees. Random forests have the following desirable features:

- They do not over-fit as more trees are added, producing an upper bound for the generalization error. The two factors involved in the generalization error are the strength of the individual classifiers in the forest and the correlation between them.
- They are relatively robust to outliers and noise.
- They are faster than other ensemble methods, such as bagging or boosting. Additionally, they are computationally simple and can be easily parallelized.
- Several incremental versions have been proposed [2, 86, 97, 119]. These implementations are expected to reduce the time to recompute the forest after new data is added to the training set.

Predictive performance measurement We have chosen four different statistics: accuracy, sensitivity, specificity and negative predictive value (NPV) defined as follows:

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$,
- $Sensitivity = \frac{TP}{TP+FN}$,
- $Specificity = \frac{TN}{FP+TN}$, and
- $NPV = \frac{TN}{FN+TN}$,

TP is the amount of true positives, TN are the true negatives, FP are the false positives and FN are false negatives. In our context, true positives are defined as the amount of c_0 test instances labeled correctly as c_0. True negatives, false positives and false negatives are likewise defined.

Not all types of miss-classification are equally significant in our context. A high number of false positives means that the agent will not be able to avoid effectively the UTS and, thus, the performance gain provided by the classifier prediction will be very low. In case of a high number of false negatives, the agent will be prevented from exploring safe state-action pairs. Depending on the environment, this might even be more harmful than false positives, because the agent might not even be able to select those state-action pairs that lead to the goal, and thus the learning would be completely unsuccessful.

8.2 USAP for Q-Learning

The classifier-aided Undesired State-Action Prediction algorithm (USAP) is a hybrid learning scheme. A RL algorithm learns the main task, aided by a two-class classifier[1], which provides a prediction $p(s,a)$ of the probability of reaching an undesired state after taking the state-action pair (s,a). Formally, $p(s,a) = \sum_{s' \in U} P(s,a,s')$.

The whole scheme is depicted in Figure 8.1. Both the RL algorithm and the SL algorithm have the same inputs: a reward signal $R(s)$ and the state s. The reinforcement learner informs which actions are taken each time-step to the classifier, and the classifier predicts the value of $p(s,a)$ for the current state s and each action $a \in A$. The normalized values of the state variables and actions are the attributes of the training instances.

Modified Q-learning algorithm We propose a modified version of the standard Q-Learning specified in Algorithm 8.1. Before running the action selection policy, the RL agent receives the predicted $p(s,a)$ for each available action $a \in A$ in the observed state s, which are used to inhibit the agent from taking risky actions when applying an action selection policy like the one specified as Algorithm 8.2, which is a modified version of the typical $\epsilon - greedy$. Actions with a probability higher or equal to 0.5 are discarded, and the typical $\epsilon - greedy$ policy is used to select an action from the rest.

After action a_t is chosen in time-step t, reward r_t is observed and the classifier is retrained with a new input $< x_{t-1,1}, \ldots, x_{t-1,|X|}, a_t, \omega(r_t) >$, where X is set of state variables spanning states in S, and attributes $x_{t,i}$, $i = 1, \ldots, \mid X \mid$ correspond to the values of these state variables.

Minimum classifier performance Before the classifier can effectively provide useful values of $p(s,a)$, it must be trained to reach a minimum performance on the task. Poor classifier performance could severely hinder the exploration, even preventing the agent from learning the task at hand. We have heuristically determined the minimum number of training instances μ the classifier needs for each action and transition class. We denote $\mu_{a,\omega}$ as the number of training instances the classifier has been trained with for action a and transition class ω. If,

[1]For simplicity, we will refer to regular classifiers, but multi-classifiers can also be used

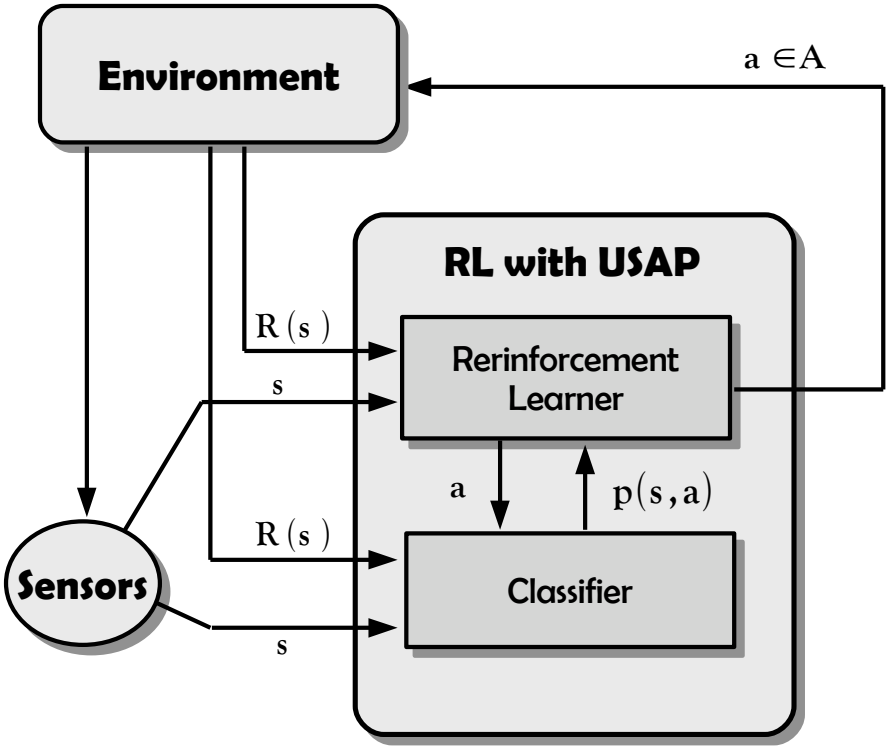


Figure 8.1: Scheme of the classifier-aided USAP.

Algorithm 8.1 Modified single agent Q-learning

Initialize $[Q_0(s,a); s \in S; a \in A]$ arbitrarily
Repeat (for each episode) n:

- Observe current state s_t
- Compute $p(s,a)$ for each $a \in A$
- Select action a_t such that $p(s_t, a_t) < 0.5$ and execute it
- Observe reward r_t and new state s'
- Retrain the classifier with instance $< s_t, a_t, \omega(r_t) >$
- Update the Q-value
  - $Q_n(s,a) = (1-\alpha_n) Q_{n-1}(s,a) + \alpha_n \left[r_t + \gamma \max_b Q_{n-1}(s', b) \right]$ if $s = s_t$; $a = a_t$
  - $Q_n(s,a) = Q_{n-1}(s,a)$ otherwise

Algorithm 8.2 Modified ϵ-greedy policy

$$\pi_\epsilon^{USAP}(s,a) = \begin{cases} 0 & if\ p(s,a) > 0.5 \\ \frac{\epsilon}{|A|} & a \neq \arg\max_{a' \in A} \{Q(s,a')\} \\ 1-\epsilon & \text{otherwise} \end{cases} \tag{8.1}$$

for a given state-action pair (s,a), $\mu_{a,c0} < \mu$ or $\mu_{a,c1} < \mu$, then $p(s,a) = 0$. This assures that no state-action is vetoed before a minimum classifier performance is acquired.

8.2.1 Distributed multi-agent implementation

Under the same assumptions for the underlying markovian process we made in Chapter 6, this RL scheme could be extended to Stochastic Games using D-RR-QL: first, agents take actions in turns following $\delta(t)$ to learn the Q-values of the C-RR-SG, and then a joint-policy can be composed using a message-based local value maximization. Distributed implementations can alleviate the communication requirements of the system and the curse of the dimensionality. Because during the learning process agents take action in turns, other agents' actions need not be an input to the classifier or the RL algorithm.

The modified D-RR-QL is shown in Algorithm 8.3. Before selecting an action, those actions such that $p(s,a)$ are discarded, and after observing a reward, a new training instance $< s_t, a_t, \omega(r_t) >$ is used to retrain the classifier. Note that only r_t is used to determine the class of state-action pair (s_t, a_t).

Algorithm 8.3 Local estimation of the i-th agent's Q^i function by the communication-free D-RR-QL algorithm with USAP.

Initialize $\left[Q_0^i(s,a) = 0; s \in S; a \in A; i = 1 \ldots N\right]$ 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, such that $p(a_t, s_t) < 0.5$
- Give turn, allowing a cycle of actions
- Observe following rewards $r_t, \ldots, r_{t+N-1}$ and new state s_{t+N}
- Retrain the classifier with instance $< s_t, a_t, \omega(r_t) >$
- Learn the state-action value matrix
  - if $s_t = s$, $a_t = a$, $i = \delta(t)$ then

$$Q_n^i(s,a) = (1-\alpha_n) Q_{n-1}^i(s,a) + \alpha_n \left[\sum_{k=0}^{N-1} \gamma^k r_{t+k} + \gamma^N \max_{a'} Q_n^i(s_{t+N}, a')\right]$$

  - $Q_n^i(s,a) = Q_{n-1}^i(s,a)$ otherwise

until s_{t+N} is terminal

This is because our goal is to avoid only those state-actions that result in an *immediate* negative reward.

8.3 Experiments

We have conducted experiments on the hose transportation task using the accurate GEDS model, and the multi-agent taxi domain (Appendix B) for the empirical measurement of the learning performance improvement using the USAP approach.

8.3.1 Off-line classification performance

This first experiment was aimed to determine which classifier should be used in the on-line experiments. To that end, agents first explored both problem domains randomly for a short amount of time. The logs of the simulation were the source to build the train and test data used to measure the predictive performance of 11 different classifiers:

- 1-NN
- 3-NN
- Naive Bayes
- Random Forest
- Naive Bayes tree
- C4.5 tree
- Clustering k-means
- Clustering Cobweb
- Clustering Exp. Max.
- $\nu-$SVM RBF and
- $\nu-$ SVM Sigmoidal.

Training and testing was performed in the Weka machine-learning environment [79]. Accuracy, sensitivity, specificity and NPV were in all cases averaged after using 10 times 10 cross-fold validation.

8.3.1.1 Hose transportation on the GEDS model

Two agents control two robots ($N = 2$) exploring randomly the state space of the hose transportation task using the GEDS model for a total of 1573 execution cycles. All the state variables defined in Appendix B were used. For each hose segment, 2 flags describe the hose configuration. Agents followed predefined turns, as proposed in the C-RR-SG framework and each agent logged its own set of training instances. Table 8.1a shows the classifier performances measured for the database generated by the first agent. The cross-validation results obtained from the second agent data are shown in Table 8.1b. From the four different predictive performance statistics we measured, Random Forest outperformed all the rest of classifiers in accuracy, sensitivity and NPV. Regarding specificity, RF offered the second best performance in both cases.

8.3.1.2 Multi-agent taxi problem

The same procedure was carried out in the taxi task: the two agents randomly explored the environment following turns during a single episode (2002 cycles of execution). Afterward, the eleven classifiers were trained with the generated training set to measure the offline predictive performance. Table 8.2a shows the cross-validation results for the first agent data, and Table 8.2b shows the results for the second agent data. In this case, the $C4.5$ tree algorithm outperforms all the other classifiers in all performance measures. In general, tree-based methods obtain better results than any of the other families of classifiers. Our

Classifier	Accuracy	Sensitivity	Specificity	NPV
1-NN	97.88	99.29	80.20	90.38
3-NN	96.40	98.95	64.40	85.23
Naive Bayes	87.78	92.05	34.11	21.71
Random Forest	**98.30**	**99.68**	81.01	**96.94**
Naive Bayes Tree	97.55	99.29	75.72	88.46
C4.5 Tree	97.30	99.34	71.75	92.13
Clustering k-means	51.95	55.52	12.31	2.91
Clustering Cobweb	61.64	67.98	16.94	3.29
Clustering Exp. Max.	60.22	60.07	62.10	10.96
$\nu-$SVM RBF	97.32	98.93	77.11	84.11
$\nu-$ SVM Sigmoidal	43.36	40.08	**84.52**	10.09
Gaussian RBF Network	92.45	99.22	7.46	45.45

(a) Cross-validation results for the first agent's data.

Classifier	Accuracy	Sensitivity	Specificity	NPV
1-NN	97.31	98.96	78.83	87.17
3-NN	94.99	98.41	56.67	76.16
Naive Bayes	87.48	91.35	44.18	31.40
Random Forest	**98.35**	**99.72**	83.02	**96.39**
Naive Bayes Tree	98.03	99.71	79.34	96.03
C4.5 Tree	98.05	99.59	80.91	94.60
Clustering k-means	58.05	60.69	28.58	06.11
Clustering Cobweb	64.38	71.79	20.66	06.13
Clustering Exp. Max.	63.89	61.53	**90.11**	17.35
$\nu-$SVM RBF	96.72	98.44	77.54	81.61
$\nu-$ SVM Sigmoidal	41.02	37.71	84.04	10.70
Gaussian RBF Network	91.77	99.42	06.29	49.34

(b) Cross-validation results for the second agent's data.

Table 8.1: Offline predictive performance on the hose transportation task using the GEDS model.

educated guess is that tree-based methods outperform other methods because the environment presents a clear structure in which attribute-based splits of the state space are able to effectively separate the state space between classes c_0 and c_1.

8.3.2 On-line classification performance

In this second experiment, we evaluate the on-line learning performance improvement of the USAP approach. So as to keep a balanced training set, a maximum of 100 instances were allowed for each combination of action-class. Whenever this limit was reached, a random sample was substituted with the new one.

In order to predict the undesired state-actions, we selected the classifier with the highest off-line score in each environment: Random Forest for the hose transportation task using the GEDS model, and C4.5 for the multi-agent taxi problem. For Random Forest, we used $log_2 |X| + 1$ attributes at each node split.

8.3.2.1 GEDS hose transportation

In this experiment, we purposefully selected no goal position and let the agents explore the environment. Three different action selection policies were tested: pure random exploration (PRE), state-action vetoes (SAV) and the modified $\pi_\epsilon^P(s,a)$ policy (USAP):

- PRE: each time-step a random action is selected.
- SAV: each time-step a random action from $A^e(s)$ is selected. The safe action repertoire is learnt using the same state representation used by the Q-Learning algorithm.
- USAP: each time-step a random action a, such that $p(s,a) < 0.5$, is selected.

In Figure 8.2 we have plotted the number of valid states visited during the exploration of the state-action space versus the number of simulation steps. The higher the number of valid states visited, the higher the probability to reach any predefined goal. Clearly, vetoing undesired state-action pairs already visited (SAV) makes exploration more efficient even if the whole set of state of variables is used to characterize the state. Nevertheless, the prediction/generalization capability of the USAP approach outperforms both PRE and SAV.

8.3.2.2 Multi-agent taxi problem

We compare the learning performance in this multi-agent scenario using D-RR-QL with and without USAP. The task was randomly explored at first ($\epsilon = 1$) and, each episode, the exploration parameter ϵ was decreased $\Delta\epsilon = -0.002$ until $\epsilon = 0$. Each 10 episodes, the learnt policy was greedily evaluated. We set the learning parameters $\alpha = 0.4$ and $\gamma = 0.9$.

Method	Accuracy	Sensitivity	Specificity	NPV
1-NN	85.9	90.4	70.88	68.86
3-NN	83.36	91.01	57.83	65.83
Naive Bayes	77.23	95.69	15.56	52.02
Random Forest	99.5	99.83	98.41	99.42
Naive Bayes Tree	98.96	99.57	96.9	98.56
C4.5 Tree	**99.89**	**99.98**	**99.56**	**99.95**
Clustering k-means	59.88	63.22	48.71	28.41
Clustering Cobweb	12.11	58.93	69.67	34.25
Clustering Exp. Max.	58.01	58.69	55.72	28.79
$\nu-$SVM RBF	97.15	97.99	94.34	93.36
$\nu-$ SVM Sigmoidal	48.42	43.56	64.72	25.56
Gaussian RBF Network	81.25	92.99	42.1	64.24

(a) Cross-validation results for the first agent's data.

Method	Accuracy	Sensitivity	Specificity	NPV
1-NN	85.340	90.07	70.00	90.82
3-NN	52.84	90.59	57.29	87.49
Naive Bayes	78.05	95.42	20.77	79.88
Random Forest	99.41	99.77	98.22	99.46
Naive Bayes Tree	99.85	99.51	96.67	99.00
C4.5 Tree	**99.99**	**99.99**	**100**	**100**
Clustering k-means	59.39	62.47	49.22	80.21
Clustering Cobweb	11.63	60.44	70.70	83.27
Clustering Exp. Max.	56.89	57.00	56.52	81.21
$\nu-$SVM RBF	96.77	97.66	93.84	98.12
$\nu-$ SVM Sigmoidal	51.08	47.66	63.73	81.11
Gaussian RBF Network	79.87	47.24	36.84	82.90

(b) Cross-validation results for the second agent's data.

Table 8.2: Offline predictive performance on the multi-agent taxi problem.

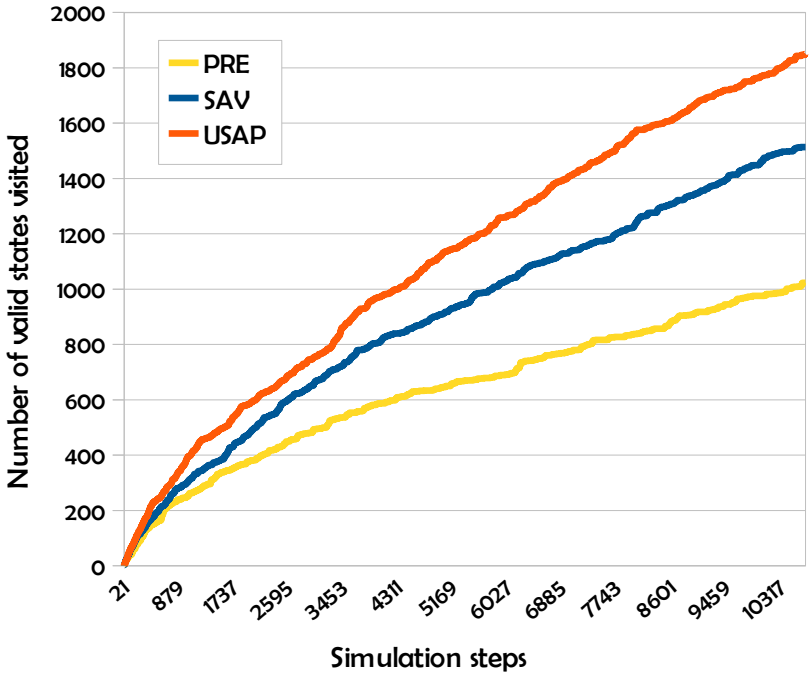


Figure 8.2: Hose transportation task with GEDS model: on-line predictive performance. Number of valid states visited.

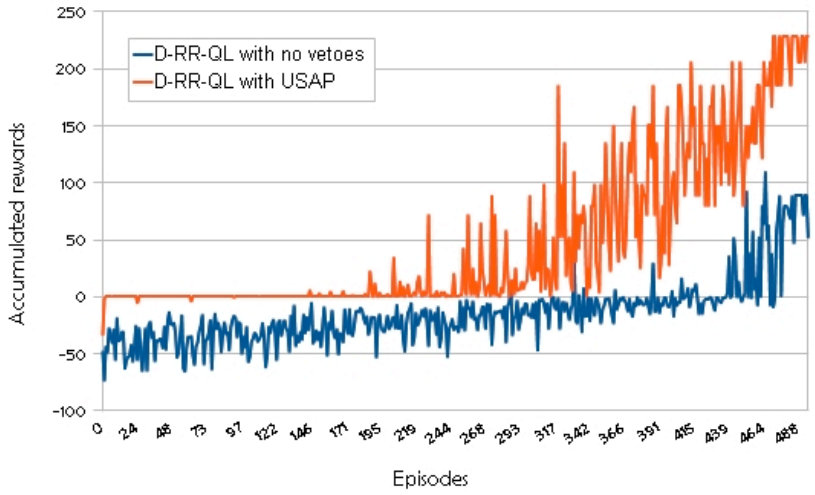
Figures 8.3a and 8.3b show the accumulated rewards and the steps needed to reach the goal during the exploration episodes, respectively. In Figure 8.3a becomes clear that the classifier is able to learn how to avoid undesired state-action pairs leading to negative rewards from the beginning. Because of this, accumulated rewards are only occasionally negative with USAP, while the bare D-RR-QL is only able to receive positive rewards at the very end of the learning process. USAP provides consistently higher accumulated rewards during all the learning process but this is not only because negative rewards due to collisions are avoided, but because focusing on states in T makes the learning more efficient. This can be seen in Figure 8.3b: the number of steps needed to reach the goal in each episode is smaller using USAP.

Figure 8.4 shows the results of the greedy evaluation episodes. D-RR-QL with USAP is able to learn a greedy policy completing the task in less than half the steps needed if USAP is not used. Not only that, the number of steps needed by these greedy policies is always smaller using USAP, and learned much faster.

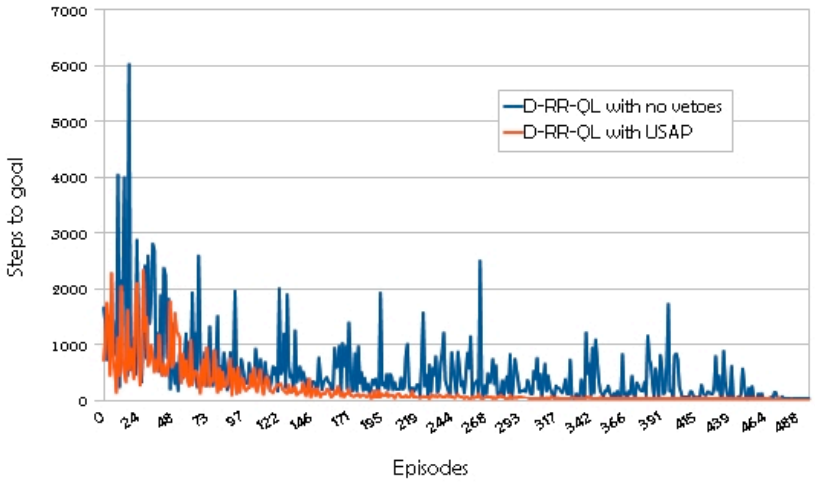
8.4 Conclusions

We have presented a hybrid classifier-aided approach to Reinforcement Learning. The learning task is decomposed into two sub-tasks: learning the main task, and predicting which state-actions may result in undesired transitions. Classifiers are able to predict the desirability of yet unseen transitions and, thus, are expected to outperform those approaches based on avoiding already experienced state-actions. We have also extended the single-agent algorithm to SG by means of using D-RR-QL.

Off-line and on-line results for computational experiments on the GEDS hose transportation task and the classical multi-agent problem report a great learning performance gain. We believe this to be a promising line of work towards RL in over-constrained environments.



(a) Multi-agent taxi problem. Accumulated rewards during the learning of the task with and without USAP.



(b) Multi-agent taxi problem. Steps per episode with and without USAP.

Figure 8.3: Multi-agent taxi problem. On-line predictive performance during the exploration phase.

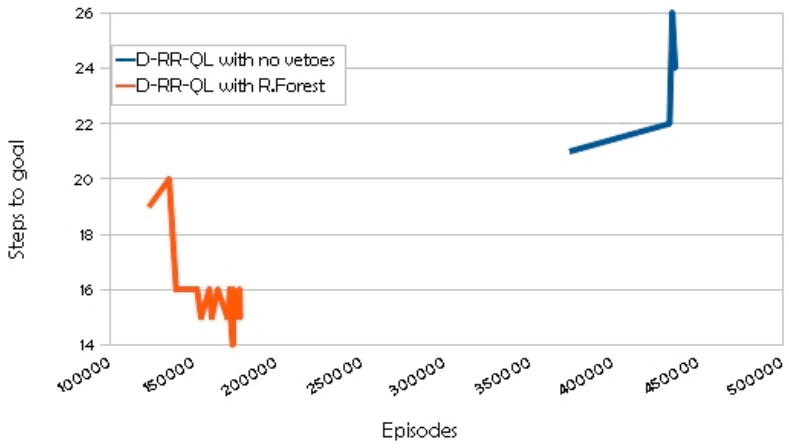


Figure 8.4: Multi-agent taxi problem. On-line greedy evaluation of learnt policies.

Appendix A

The geometrically exact dynamic splines model

In this appendix, we review the Geometrically Exact Dynamic Splines (GEDS) model which has been used for the accurate modeling of the hose dynamics to support the simulations performed to show the application of the MARL algorithhms discussed in this Thesis to learn the multi-robot control algorithms for the hose manipulation [36, 35]. The goal of this modeling is the dynamical interaction between hose and robots, how the motion and forces of the robots modify the hose spatial configuration to be able to compute the effect of the control commands.

The combination of spline geometrical modeling and physical dynamical models was introduced by [92]. They allow a continuous definition of uni-dimensional objects. An inconvenient of the spline model is that, since it is exclusively based on the spline control points, it is unsuitable for representing the hose torsion. The work of [116] has improved the spline representation by combining the spline-based modeling with the Cosserat rod theory [5, 96], allowing to model the twisting of the hose. This approach, known as Geometrically Exact Dynamic Splines (GEDS), represents the control points of the splines by the three Cartesian coordinates plus a fourth coordinate representing the twisting state of the hose.

A.1 Geometry of the hose

The Cosserat rod theory [96, 5] is usually used in modeling uni-dimensional objects because it permits to model its physical behavior. In Cosserat rod theory an uni-dimensional object is described by a curve $\mathbf{r}(u)$ and a coordinate frame of director vectors $[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3](u)$ attached to each point of the curve. The parameter u goes from one end of the curve, for $u = 0$, to the other for $u = L$, being L the length of the hose. The curve and the director vectors are joined

into a coordinate frame $E(u) = [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{r}](u)$. A graphic representation of the hose by the curve and the frame director vectors is shown in figure A.1.

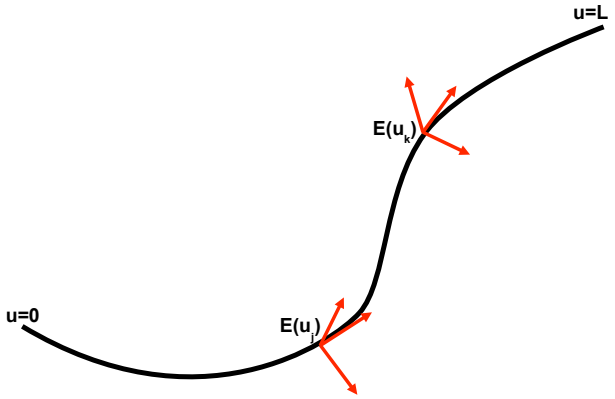


Figure A.1: Cosserat rod model of a hose.

The GEDS describe the uni-dimensional object by a spline model taking into account the Cosserat rod approach in order to model the twisting behavior of the hose. A spline is a piecewise polynomial function. See figure A.2 for an illustration. Splines define a curve by means of a collection of Control Points, which define a function that allows to compute the whole curve.

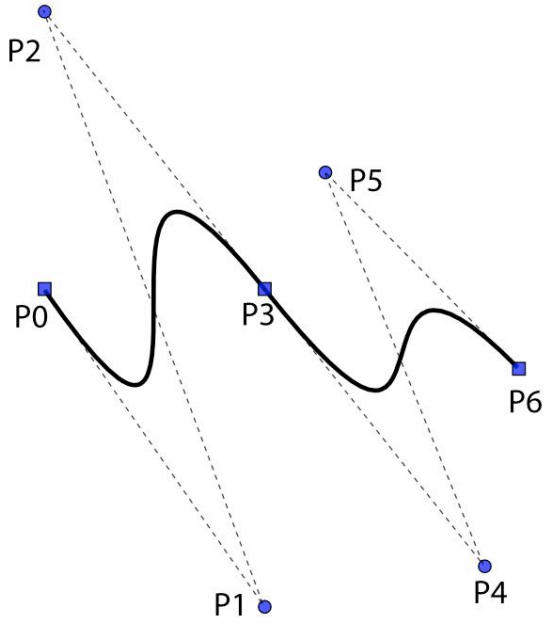


Figure A.2: Cubic spline.

The spline expression for a curve $\mathbf{q}(u)$ is a linear combination of control points $\mathbf{p}_i$ where the linear coefficients are the polynomials $N_i(u)$ which depend on the parameter u defined in $[0, 1)$. The spline equation is:

$$\mathbf{q}(u) = \sum_{i=0}^{n} N_i(u) \cdot \mathbf{p}_i, \tag{A.1}$$

where $N_i(u)$ is the polynomial associated to the control point $\mathbf{p}_i$, and $\mathbf{q}(u)$ is the point of the curve at the parameter value u. It is possible to travel over the curve by varying the value of parameter u, starting at one end for $u = 0$ and finishing at the other end for $u = 1$. We use a B-spline model for the hose, requiring a set of control points, a set of knots and a set of coefficients, one for each control point, ensuring that all curve segments are joined together satisfying certain continuity conditions.

Given $n+1$ control points $\{\mathbf{p}_0, \ldots, \mathbf{p}_n\}$ and a knot vector $\mathbf{U} = \{u_0, \ldots, u_m\}$, the B-spline curve of degree p defined by these control points and knot vector $\mathbf{U}$ is:

$$\mathbf{q}(u) = \sum_{i=0}^{n} N_{i,p}(u) \cdot \mathbf{p}_i, \tag{A.2}$$

where $N_{i,p}(u)$ are B-spline basis functions of degree p ($p = 3$ in this work), built using the Cox de Boor's algorithm.

$$\begin{aligned} N_{i,0}(u) &= \begin{cases} 1 & u_i \leq u < u_{i+1} \\ 0 & c.c. \end{cases}, \\ N_{i,p}(u) &= \frac{u - u_i}{u_{i+p} - u_i} \cdot N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} \cdot N_{i+1,p-1}(u). \end{aligned}$$

Because the control points of the curve will vary in time, we rewrite equation (A.2) in terms of the time parameter t:

$$\mathbf{q}(u, t) = \sum_{i=0}^{n} N_{i,p}(u) \cdot \mathbf{p}_i(t). \tag{A.3}$$

This extended model receives the name of *Dynamic splines,* and it is the model that we have used to model the hose. If we want to take into account the hose internal dynamics, we need also to include the hose twisting at each point given by the rotation of the transverse section around the axis normal to its center point, in order to compute the hose potential energy induced forces. In the GEDS approach, the hose model follows the Cosserat rod approach characterizing it by the curve given by the transverse section centers $\mathbf{c} = (x, y, z)$, and the orientation of each transverse section θ. This description is summarized by the following notation: $\mathbf{q} = (\mathbf{c}, \theta) = (x, y, z, \theta)$. In figure A.3, the relation between the Cosserat rod director vectors and the twisting angle θ is shown, where vector $\mathbf{t}$ represents the tangent to the curve at point $\mathbf{c}$, and vectors $\mathbf{n}$ and $\mathbf{b}$ determine the angle θ of the transverse section at point $\mathbf{c}$.

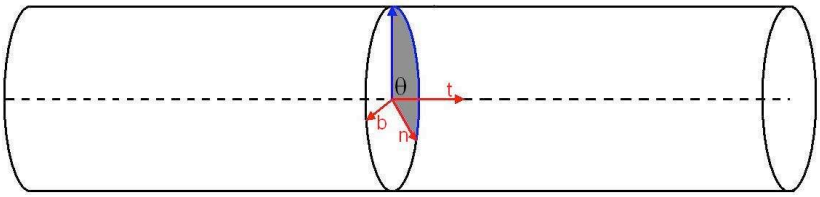
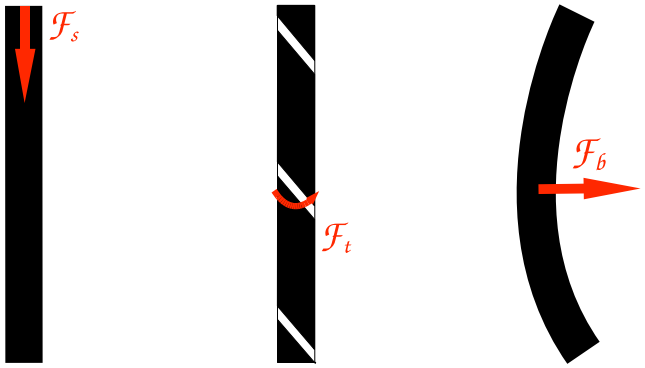


Figure A.3: Hose section.



(a) Stretching force. (b) Twisting force. (c) Bending force.

Figure A.4: Forces induced by the potential energy of the hose.

A.2 Hose dynamical model

From the Cosserat representation and applying the Lagrange equation (A.4) a mathematical relation between the potential energy $\mathcal{U}$, the Kinetic energy $\mathcal{T}$ and the generalized external forces $\mathbf{F}$ is obtained.

$$\frac{d}{dt}\left(\frac{\partial \mathcal{T}}{\partial \dot{\mathbf{p}}_i}\right) = \mathbf{F}_i - \frac{\partial \mathcal{U}}{\partial \mathbf{p}_i}. \tag{A.4}$$

The kinetic energy is the motion energy, while the potential energy is the energy due to the hose configuration. Let $\mathbf{F} = \{\mathbf{F}_0, \mathbf{F}_1, \ldots, \mathbf{F}_n\}$ denote the model of the external forces acting on the hose spline model control points. Each $\mathbf{F}_i$ acts on control point $\mathbf{p}_i$. It is usually assumed that mass and stress are homogeneously distributed among the $n+1$ degrees of freedom of the hose spline control model.

A.2.1 Potential Energy

It is necessary to determine the forces that will be generated on the hose as a consequence of its potential energy due to its physical configuration.

In figure A.4 we can appreciate the forces and torques $\mathbf{F}_U = (\mathbf{F}_s, \mathbf{F}_t, \mathbf{F}_b)^T$ that deform the hose because of its potential energy. The stretching force, $\mathbf{F}_s$, is the force along the normal to the hose transverse section and its application results in its lengthening. The tension torque, $\mathbf{F}_t$, makes the transverse section to rotate around the center of the section. The bending torque, $\mathbf{F}_b$, modifies the orientation of the transverse section. The forces acting on the transverse section plane are neglected, because of the Kirchhoff assumption that the transverse sections are rigid and that only the hose curvature may be distorted.

In mechanics and physics, the Hooke's law provides an approximation for linear-elastic materials. This law establishes that the extension of a spring is in direct proportion to the load applied to it. Summarizing, the Hooke's law for a spring-mass system establishes:

$$F = -kx, \tag{A.5}$$

where x is the displacement of the spring due to the load applied to it, k is the spring constant and F the restoring force experimented by the spring due to its material properties. In general the Hooke's law is applied to elastic materials because their behavior is similar to the spring as its molecules return to the initial state of stable equilibrium, quickly regaining the object its original shape after a force has been applied.

Let us denote the length of the hose as L, and the area of the transverse section as A, then the hose length extension is linearly proportional to the deformation resistance of the hose:

$$\triangle L = \frac{F}{EA} L, \tag{A.6}$$

where E is the modulus of elasticity, which is the mathematical description for the hose resistance to be deformed when a force is applied to it.

Isolating the value of F in equation A.6 we have:

$$F = EA \frac{\triangle L}{L}. \tag{A.7}$$

Defining ϵ as the deformation of the hose relative to the transverse area, $\epsilon = A\frac{\triangle L}{L}$, we can rewrite A.7 as:

$$F = E\epsilon, \tag{A.8}$$

which is a version of the Hooke's law for an elastic uni-dimensional object.

When a small deformation is considered for a relative big radius of the hose length in comparison with the radius of the transverse section, it is said that the hose is in a linear elasticity dynamic regime, and then the force equation A.8 may be applied for each of the stretching, twisting and bending forces. The matricial version for the stretching, twisting and bending forces is:

$$F_U = H\epsilon = \begin{pmatrix} E_s & 0 & 0 \\ 0 & E_t & 0 \\ 0 & 0 & E_b \end{pmatrix} \cdot \epsilon. \tag{A.9}$$

The deformation vector, $\epsilon = (\epsilon_s, \epsilon_t, \epsilon_b)^T$, is composed of the stretching deformation ϵ_s, the twisting deformation ϵ_t and the bending deformation ϵ_b. The Hooke matrix, H, is composed of the stretching rigidity E_s, the twisting rigidity E_t and the bending rigidity E_b.

Maintaining the spring-mass system analogy, the potential energy U is defined as $U = \frac{1}{2}kx^2$, that in the case of the hose is defined by the following integration from $u = 0$ up to $u = L$:

$$U = \frac{1}{2}\int_0^L \epsilon^T \mathbf{F}_U du. \tag{A.10}$$

Using the definition of $\mathbf{F}_U$ from equation A.9 in equation A.10 we have:

$$U = \frac{1}{2}\int_0^L \epsilon^T H \epsilon du$$

Note that this model is appropriated for a hose that in rest configuration is stiffed and not twisted or bended, but for a cable as a telephone cord or a spring the rest configuration of the hose is different to zero, so ϵ should be replaced by $(\epsilon - \epsilon_0)$, being ϵ_0 the rest strain.

When the objects lie in fact in the 2D space we can obviate the moments. Therefore, the potential energy $\mathcal{U}$ is defined by the following integration along the hose, from $u = 0$ up to $u = L$:

$$\mathcal{U} = \frac{1}{2}\int_0^L (\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_0)^T \mathbf{F}_{\mathcal{U}} du, \tag{A.11}$$

where $\mathbf{F}_{\mathcal{U}} = (\mathcal{F}_s, \mathcal{F}_t, \mathcal{F}_b)^T$ are, respectively, the stretching force, and the torsion and bending moments suffered by the hose due to its configuration, $\boldsymbol{\epsilon} = (\epsilon_s, \epsilon_t, \epsilon_b)^T$ is the deformation vector.

A.2.2 Kinetic energy

The kinetic energy T is composed of the translation energy T_t and the rotation energy T_r.

$$T = T_t + T_r. \tag{A.12}$$

The kinetic energy is given by:

$$T_t = \frac{1}{2}\mu A \int_0^L \dot{\mathbf{q}}^2 du, \tag{A.13}$$

$$T_r = \frac{1}{2}\mu \int_0^L \boldsymbol{\Omega}^T I \boldsymbol{\Omega} du, \tag{A.14}$$

where A is the area of the transversal section,$\boldsymbol{\Omega}$ is the angular velocity, μ is the linear density and I is the polar momentum of inertia.

A simplified version of the kinetic energy expression is given by defining the inertial matrix J, which is invariant over all the hose points because of the assumption of constant hose diameter.

$$J = \begin{pmatrix} \mu & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 \\ 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & I \end{pmatrix}.$$

The kinetic energy of the hose T is then defined by:

$$T = \frac{1}{2}\int_0^L \frac{d\mathbf{q}^T}{dt} J \frac{d\mathbf{q}}{dt} du. \tag{A.15}$$

A.2.3 Dynamic model

The kinetic energy model takes into account the translational and rotational motions of the hose, therefore, we can determine from it the acceleration of every hose point, by deriving equation A.15. The left hand term of equation A.4 becomes:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\mathbf{p}}_i}\right) = \frac{1}{2}\int_0^L \frac{d}{dt}\frac{\partial\left(\dot{\mathbf{q}}^T J \dot{\mathbf{q}}\right)}{\partial \dot{\mathbf{p}}_i} du. \tag{A.16}$$

Next, we consider the derivative of the potential energy relative to a generalized coordinate:

$$\frac{\partial U}{\partial \mathbf{p}_i} = \frac{1}{2}\int_0^L \frac{\partial \epsilon^T H \epsilon}{\partial \mathbf{p}_i} du. \tag{A.17}$$

The aim of the physical modeling is to determine the accelerations of the hose, in terms of its geometrical model, therefore the accelerations are obtained in the GEDS model substituting $\mathbf{q}$ in the expression of equation A.16 by the right side of equation A.3:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{\mathbf{p}}_i} = \sum_{j=0}^{n} J \frac{d^2\mathbf{p}_j}{dt^2}\int_0^L (N_i(u)N_j(u))du \tag{A.18}$$

Defining:

$$\mathbf{M}_{ij} = J\int_0^L (N_i(u)N_j(u))du$$

and

$$\mathbf{A} = \left[\frac{d^2\mathbf{p}_j}{dt^2}\right],$$

The Lagrange equation (equation A.4) becomes:

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{\mathbf{p}}_i} = \sum_{j=0}^{n} \mathbf{M}_{i,j}\mathbf{A}_j. \tag{A.19}$$

Using equations A.19 and A.17 the Lagrange equation is written in a matrix form:

$$\mathbf{MA} = \mathbf{F} + \mathbf{P}, \tag{A.20}$$

where $\mathbf{P} = \left[\frac{\partial U}{\partial \mathbf{p}_i}\right]$.

A.3 Hose-robots dynamical interaction model

The whole system model, composed by the robots and the hose-like linking element, is built from the uni-dimensional element GEDS model by specifying the positions u_r of the robots along the hose. A configuration h of the hose-multi-robot system is defined as:

$$h = \{\mathbf{p}, \mathbf{U}, \mathbf{U_r}\}, \tag{A.21}$$

where

- $\mathbf{p}$ is the control point vector of the hose B-spline model,
- $\mathbf{U}$ is the collection of knots in the B-spline model,
- $\mathbf{U_r} \subset \mathbf{U}$ is the collection of knots that correspond to robot attachments to the hose.

The robot knot vector $\mathbf{U_r} = \{u_{r_i}\}$ contains the values of the arc length parameter u where the robots are attached to the hose. The spatial position of the i-Th robot $\mathbf{r}_i$ is given by the expression:

$$\mathbf{q}(u_{r_i}) = \sum_{i=0}^{n} N_i(u_{r_i}) \cdot \mathbf{p}_i = \mathbf{r}_i.$$

The information we have about the hose is a sequence τ of sampling points of the hose center curve, containing the Cartesian position of every point (x, y, z) and its torsion angle θ. Then, we construct the initial hose configuration h_0 by an interpolation method that generates the control points of the B-spline cubic curve that interpolates this points.More precisely, we make an uniform sampling of the hose center curve to obtain the sequence of points τ in order to get a uniform B-spline interpolating. This distribution of the sampling points optimizes the performance of the IFBA, avoiding the occurrence of spurious peaks, protuberances and loops. The sampling is done taking into account the number of knots we want to use, depending on the relation between precision and computing time that we desire.

The uniform selection of the interpolating points τ is obtained by dividing the hose length into n segments, being $n + 1$ the number of control points, and

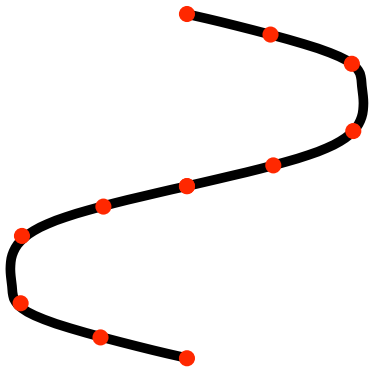


Figure A.5: Uniform selection of the interpolating points.

choosing for each division point the hose point closest to it. Figure A.5 shows the hose in black and the selected interpolating points in red, for a number of control points $n = 10$.

Equation A.20 relates the acceleration at the control points with the internal energy of the hose and the external forces applied to it. Among the external forces $\mathbf{F}$ that act on the control points, we differentiate those resulting from the ones applied by the robots $\mathbf{F}_p$ from other external forces $\mathbf{F}_e$:

$$\mathbf{F} = \mathbf{F}_p + \mathbf{F}_e. \tag{A.22}$$

The relation between the forces applied on the robot attaching points $\mathbf{F_r}$ and the resulting forces on the control points $\mathbf{F_p}$ is given by:

$$\mathbf{F_p} = J_{pr} \cdot \mathbf{F}_r, \tag{A.23}$$

on the basis of the Jacobian matrix J_{pr} relating robot positions and control points:

$$J_{pr} = \begin{pmatrix} \frac{\partial \mathbf{q}(u_{r_1})}{\partial \mathbf{p}_0} & \cdots & \frac{\partial \mathbf{q}(u_{r_l})}{\partial \mathbf{p}_0} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{q}(u_{r_1})}{\partial \mathbf{p}_n} & \cdots & \frac{\partial \mathbf{q}(u_{r_l})}{\partial \mathbf{p}_n} \end{pmatrix} = \begin{pmatrix} N_0(u_{r_1}) & \cdots & N_0(u_{r_l}) \\ \vdots & \ddots & \vdots \\ N_n(u_{r_1}) & \cdots & N_n(u_{r_l}) \end{pmatrix}, \tag{A.24}$$

So, we rewrite equation A.20 in order to determine the forces that the robot must exert on the hose attachment points as a function of the desired accelerations on the control points, the external forces on the hose and the energy configuration:

$$\mathbf{F}_p = M\mathbf{A} - \mathbf{F}_e - \mathbf{P}. \tag{A.25}$$

Since the hose dynamics are defined on the control points $\mathbf{p}_i$, a force $\mathbf{f}$ applied on a particular point of the hose is decomposed into the forces $\mathbf{f}_i$ resulting at

each spline control point $\mathbf{p}_i$. The partial derivative of a point $\mathbf{q}(u)$ in the curve respect to the control point $\mathbf{p}_i$ is:

$$\frac{d\mathbf{q}(u)}{d\mathbf{p}_i} = N_i(u). \tag{A.26}$$

For a control point $\mathbf{p}_i$, the corresponding force $\mathbf{f}_i$ is computed as:

$$\mathbf{f}_i = \mathbf{f}\frac{\partial \mathbf{q}}{\partial \mathbf{p}_i} = \mathbf{f} \cdot N_i. \tag{A.27}$$

Defining the Jacobian matrix J_{rq} of the robot contact points with the hose as a function of the control points, we have:

$$J_{pr} = \begin{pmatrix} \frac{\partial \mathbf{q}(u_{r_1})}{\partial \mathbf{p}_0} & \cdots & \frac{\partial \mathbf{q}(u_{r_l})}{\partial \mathbf{p}_0} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{q}(u_{r_1})}{\partial \mathbf{p}_n} & \cdots & \frac{\partial \mathbf{q}(u_{r_l})}{\partial \mathbf{p}_n} \end{pmatrix} = \begin{pmatrix} N_0(u_{r_1}) & \cdots & N_0(u_{r_l}) \\ \vdots & \ddots & \vdots \\ N_n(u_{r_1}) & \cdots & N_n(u_{r_l}) \end{pmatrix}, \tag{A.28}$$

where u_{r_j} is the attachment point of the robot $\mathbf{r}_j$ to the hose.

We use the Jacobian matrix J_{pr}, defined in equation A.28, to obtain the relation between the applied forces in the robot attaching points $\mathbf{F_r}$ and the resulting forces on the control points $\mathbf{F_p}$:

$$\mathbf{F_p} = J_{pr} \cdot \mathbf{F}_r. \tag{A.29}$$

Appendix B

Simulation environments

In this appendix, we describe the three different simulation environments used in this work as case studies, namely: the hose transportation task using an accurate GEDS model, the hose transportation using a simplified model, and the multi-agent taxi problem. For each environment we give detailed description of the corresponding MDP including state variables, reward signals and the set of available actions. It is important to notice that the state variables of the MDP are not used by the simulation models, in fact they are often computed from the simulation actual configuration to be used by the RL or MARL algorithm as the current observations of the real system.

B.1 Hose transportation task using the GEDS model

Figure B.1 depicts a hose-robots configuration, dots corresponding to robots. The arrows correspond to the forces exerted by the robots trying to move in some specific direction. The distinctive feature of this environment is the attempt to follow an accurate simulation of the hose dynamics using a GEDS model described in Appendix A. The task in this environment is the deployment of the hose using a collection of N robots from an arbitrary initial configuration, bringing the tip of the hose to a predetermined goal position, while the other end remains attached to a source position. The robotic units are uniformly distributed along the length of the hose. Agents must control robots so that the tip of the hose (carried by robot P_1) reaches a goal position G. The experimental environment is a $1m \times 1m$ grid, which is perceived as a grid of 20×20 cells by the agents. Given any position $P \in \mathbb{R}^2$ on the working space, a discretization function $c : \mathbb{R}^2 \to \mathbb{Z}^2 \in [20 \times 20]$, $c(P)$ provides the integer coordinates of the cell where P lies. The following constraints are defined in the simulation:

- Robots cannot not drive over the hose because they will get stuck.
- Hose segments between robots cannot be stretched over their nominal

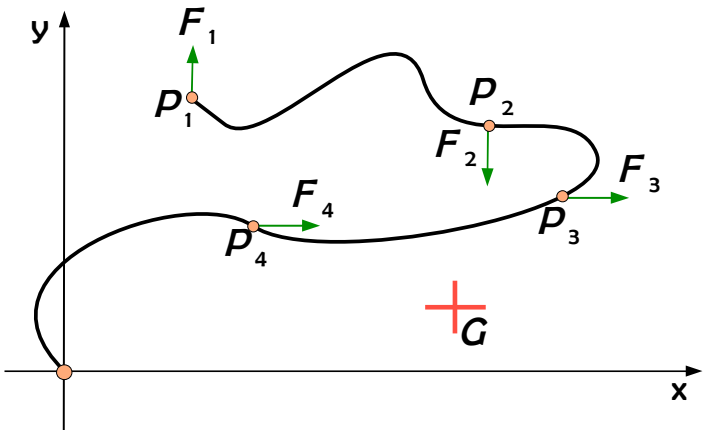


Figure B.1: GEDS hose transportation problem. One end of the hose is attached to the source, the tip of the hose is driven by a robot. Other robots are attached at the middle sections of the hose to help in the deployment.

length because the hose will be broken. In the simulated system, hose length may vary because the spline approximation does not ensure a constant length. The stretching force reacts against changes in the hose's length, however if the length goes beyond an elastic limit, the simulation software reports that the hose is broken.

- Collisions between robots produce the effect of blocking them. In the physical world, robot collisions may produce entanglements and broken pieces, therefore the simulation assumes this event as a terminal event.
- All robots should stay within the boundaries of the simulated world.

B.1.1 Action space

For each robot $i = 1 \ldots N$, the set of allowed actions is defined as $A_i = \{up_i, down_i, left_i, right_i, none\}$. In MARL approaches, agent i selects an action from A_i, while in single-agent RL approaches, the agent chooses an action from the joint action set $A = \times_{i=1}^{N} A_i$. These discrete symbolic actions result in some fixed magnitude forces F_i, which are applied on the hose by each robot in the formation. The GEDS model takes these forces into account for the accurate simulation.

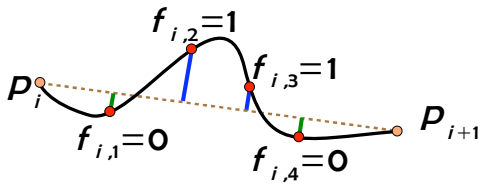


Figure B.2: Graphical representation of flags $f_{i,j}$ for a given spline between points P_i and P_{i+1}.

B.1.2 State variables

The MDP for the RL and MARL algorithmos has the following state variables:

- $c(G)$: the cell coordinates of the goal position on the grid.
- $r_i^n = c(P_{i-1}) - c(P_i)\,,\ i = 1 \dots N$: the relative position of the i^{th} robot with respect to the next robot on the hose. For practical reasons, $r_0^n = (0,0)$.
- $r_i^p = c(P_i) - c(P_{i+1})\,, i = 0 \dots N-1$: the relative position of the i^{th} robot with respect to the previous robot on the hose. For practical reasons, $r_N^p = (0,0)$.
- $f_{i,j}$, $i = 1 \dots N$, $j = 1 \dots k$: k binary flags describing the spline configuration between robots in positions P_i and P_{i+1}. These flags are represented in Figure B.2. First, k uniformly distributed points $p_{i,j}$ are derived from the equation defining the spline. Then, each $f_{i,j}$ takes value 0 or 1, depending on which side of the imaginary line between P_i and P_{i+1} lies the the point $p_{i,j}$. These flags provide an abstract simplified representation of the hose segment configuration which will be used by the RL and MARL to predict contraint breaking.

B.1.3 Reward function

The reward signal $R(s)$ is defined as follows:

- a negative constant value if the hose is rolled over by any robot ,
- a negative constant value if the length of a hose segment exceeds its maximum nominal value and is declared broken,
- a positive constant value if none of the previous happens and $P_1 = G$, and
- a null value otherwise.

B.2 Hose transportation task on a simplified model

Again the task of the agent in this environment is to learn the optimal control for N robots attached to a hose, transporting it to a predefined goal configuration. We denote the position of the i^{th} robot as $P_i, i = 1 \dots N$ and the hose is modeled as a set of ordered line segments between robots P_i and $P_{i+1}, i = 1 \dots N - 1$. The last line segment goes from the source of the hose (coordinates $(0, 0)$) and P_N. The tip of the hose is given by the position of robot P_1. The total length of the hose is noted L_{hose}, whereas each hose segment's maximal nominal length is $L_s = \frac{L_{hose}}{N}$. The robots are simulated on a rectangular grid of size $[20, 20]$ centered around the source point $(0, 0)$. All coordinates are integer corresponding to grid cell coordinates.

In our experiments, we have used two different simplified variants of the main hose transportation task: the fully-cooperative and the non-cooperative task. These two MDPs have two different reward signals: the former uses a shared reward signal that received by all agents, whereas in the latter each agent controlling a robot gets the reward depending on its own goal. The non-cooperative system needs to specify the goal position of each robot, that is the final global configuration of the system.

B.2.1 Action space

For each robot $i = 1 \dots N$, the set of allowed actions is defined as $A_i = \{up_i, down_i, left_i, right_i, none\}$. In multi-agent approaches, agent i selects an action from A_i and, in single-agent approaches, the agent chooses an action from the joint action set $A = \times_{i=1}^{N} A_i$

B.2.2 State variables

The fully-cooperative MARL works on the following state variables:

- G: the of the goal position on the grid given in the global coordinate system.
- $p_i, i = 0 \dots N$: the position of the i^{th} robot on the grid given in the global coordinate system.
- $r_i^g = G - P_{i+1}, \ i = 0 \dots N$: the position of the i^{th} robot relative to the goal position.
- $d_i^g = \mid G - P_i \mid, i = 0 \dots N$: the absolute distance from the i^{th} robot to the goal.
- $\theta_i^g = \arctan(G - P_i), \ i = 0 \dots N$: the angle formed by the virtual line segment between the i^{th} robot and the goal, discretized as an integer value in range $[0, 7]$.
- $r_i^n = P_{i-1} - P_i, \ i = 1 \dots N$: the position of the i^{th} robot relative to the next robot on the hose. For practical reasons, $r_0^n = (0, 0)$.

- $r_i^p = P_i - P_{i+1},\ i = 0 \dots N-1$: the position of the i^{th} robot relative to the previous robot on the hose. For practical reasons $r_N^p = (0, 0)$.

- $o_i^{up}, o_i^{down}, o_i^{left}, o_i^{right},\ i = 0 \dots N$: four boolean flags indicating whether an obstacle exists in the cells adjacent to the i^{th} robot within a time-step action range.

The non-cooperative MARL has these state variables:

- G_i: the absolute goal position for the i^{th} robot on the grid

- $p_i, d_i^n, d_i^p, o_i^{up}, o_i^{down}, o_i^{left}$ and o_i^{right} are defined as in the cooperative task

- $r_i^{g*} = G_i - P_{i+1},\ i = 0 \dots N$: the position of the i^{th} robot relative to the goal.

- $d_i^{g*} = \mid G_i - P_i \mid,\ i = 0 \dots N$: the distance from the i^{th} robot to its goal the goal discretized as an integer value in range $[0, 255]$.

- $\theta_i^{g*} = \arctan\left(G_i - P_i\right),\ i = 0 \dots N$: the angle formed by the virtual line segment between the i^{th} robot and the goal, discretized as an integer value in range $[0, 7]$.

B.2.3 Reward functions

Modular approaches (MSAV) use four different reward signals:

- $R_i^g\left(s\right)$: positive if all the constraint-reward signals are zero (the state is feasible) and $P_0 = G$. Zero otherwise.

- $R_i^{stretch}\left(s\right)$: negative if $\mid P_i - P_{i-1} \mid > L_s$ or $\mid P_i - P_{i+1} \mid > L_s$. Zero otherwise.

- $R_i^{out}\left(s\right)$: negative if $\mid P_i^x \mid > s_x$ or $\mid P_i^y \mid > s_y$. Zero otherwise.

- $R_i^{collision}\left(s\right)$: negative if $\exists j = 1 \dots N;\ i \neq j \wedge P_i = P_j$. Zero otherwise.

In monolithic (non-modular) multi-agent versions of the MDP, the reward signal is defined as the sum

$$R_i\left(s\right) = R_i^g\left(s\right) + R_i^{stretch}\left(s\right) + R_i^{out}\left(s\right) + R_i^{collision}\left(s\right),$$

Finally, for single-agent MDP environments the reward function is defined as the sum of the rewards received by all agents:

$$R\left(s\right) = \sum_{i=1 \dots N} R_i\left(s\right).$$

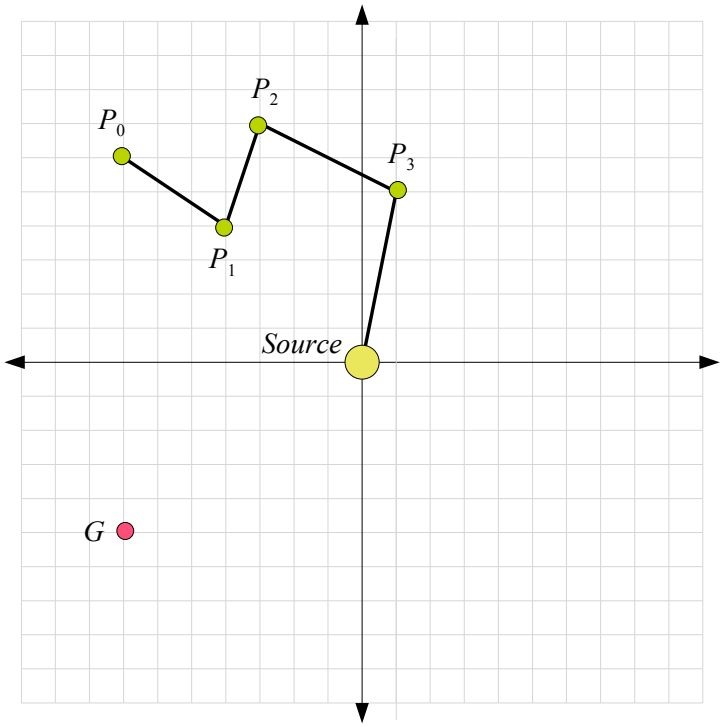


Figure B.3: The simplified hose transportation task with a unique common goal: the goal of the system is to reach point G with the tip of the hose.

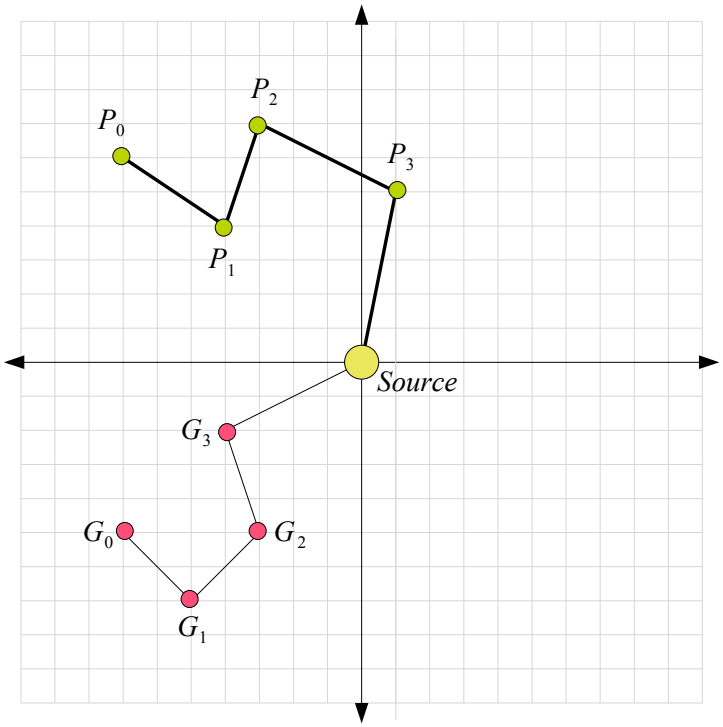


Figure B.4: The simplified hose transportation task with individual goals : the goal for each robot is to reach its own goal position G_i.

B.3 Multi-agent taxi problem

The instance of the multi-agent task we used in our experiments is an adapted version of the classical problem described in [29, 48]. Figure B.5 represents the environment. Two taxis (labeled T_1 and T_2) must carry two passengers from their initial location to their goal. In our experiments, the first passenger was initially in station R and must be carried to station B, and the second is in station G and wants to go to station Y.

B.3.1 Action space

Multi-agent environments allow agents to choose for the allotted taxi $A_i = \{up_i, right_i, down_i, left_i, pick_i, drop_i\}$. Actions $pick_i$ and $drop_i$ are used to pick the passengers at their location and drop them at the desired destination. The single-agent variant gives the agent the chance to select actions from $\mathbf{A} = \times_{i=1}^{2} A_i$.

B.3.2 State variables

The state variables assumed in the original problem are:

- The position of both taxis.
- The state of both passengers: at the initial point, on a taxi or at the goal.

To show the importance of using only relevant state variables in modular approaches, we also defined:

- $o_i^{up}, o_i^{down}, o_i^{left}, o_i^{right}$, $i = 0 \ldots N$: four boolean flags indicating whether an obstacle exists in the cells adjacent to the taxi T_i within a time-step action range.

B.3.3 Reward signals

Collisions against taxis and walls and forbidden actions (i.e., take $drop_i$ action when the taxi i is empty) get a negative reward. Once both passengers are carried to their destination, both agents receive a positive reward. A neutral reward is generated otherwise.

Modular approaches decompose this reward signal as two different components: a strictly positive reward $R^{goal}(s)$ and a strictly negative reward signal $R^{collision}(s)$.

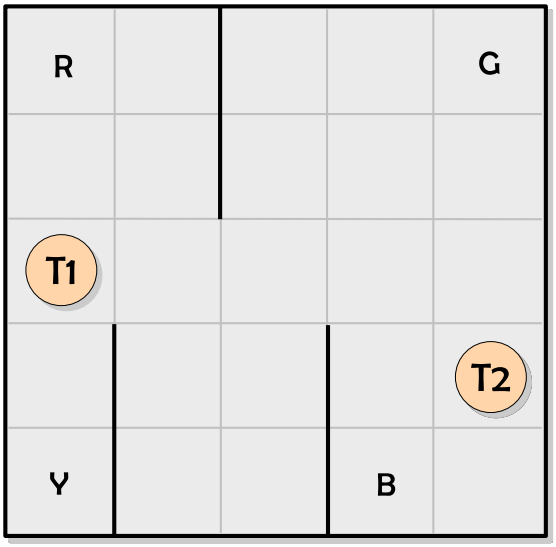


Figure B.5: Graphical representation of the multi-agent taxi problem.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *in Proc. 21st International Conference on Machine Learning*, pages 1–8. ICML, 2005.

[2] Hanady Abdulsalam, David B. Skillicorn, and Patrick Martin. Streaming random forests. In *Proceedings of the 11th International Database Engineering and Applications Symposium*, pages 225–232, Washington, DC, USA, 2007. IEEE Computer Society.

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

[4] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *In Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 119–125. AAAI Press, 2002.

[5] S. S. Antman. *Nonlinear Problems of Elasticity*. Springer-Verlag, 1995.

[6] 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.

[7] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Vision-based behavior acquisition for a shooting robot by using a reinforcement learning. In *In Proc. of IAPR/IEEE Workshop on Visual Behaviors*, pages 112–118, 1994.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Sooraj Bhat, Charles L. Isbell, and Michael Mateas. On the difficulty of modular reinforcement learning for real-world partial programming. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06, pages 318–323. AAAI Press, 2006.

[14] Maria-Iuliana Bocicor, Gabriela Czibula, and István Gergely Czibula. A distributed q-learning approach to fragment assembly. *Studies in Informatics and Control*, 20:221–232, 2011.

[15] 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.

[16] 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.

[17] 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.

[18] Leo Breiman. Random forests. *Mach. Learn.*, 45:5–32, October 2001.

[19] 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.

[20] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *IJCAI'91 Proceedings of the 12th international joint conference on Artificial intelligence*, volume 2, pages pp. 726–731. Morgan Kaufmann, 1991.

[21] Xiaobei Cheng, Jing Shen, Haibo Liu, and Guochang Gu. Multi-robot cooperation based on hierarchical reinforcement learning. *Lecture Notes in Computer Science*, 4489:90–97, 2007.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] Tom Croonenborghs, Kurt Driessens, and Maurice Bruynooghe. Learning relational options for inductive transfer in relational reinforcement learning. In *In Proceedings of the Seventeenth Conference on Inductive Logic Programming*, 2007.

[28] Gabriela Czibula, Maria-Iuliana Bocicor, and István Gergely Czibula. Solving the protein folding problem using a distributed q-learning approach. *International Journal of Computers*, 5:404–413, 2011.

[29] 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.

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

[31] 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.

[32] Yong Duan and Xin-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, 2005.

[33] R.J. Duro, M. 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.

[34] 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. Intelligent Robotics and Neuroscience.

[35] 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.

[36] Zelmar Echegoyen. *Contributions to Visual Servoing for Legged and Linked Multicomponent Robots*. PhD thesis, UPV/EHU, 2009.

[37] K. Ferguson and S. Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. In *In ICML Workshop on Transfer Learning*, 2006.

[38] B. Fernandez-Gauna, J. M. Lopez-Guede, E. Zulueta, and M. Graña. Learning hose transport control with Q-learning. *Neural Network World*, 20(7):913–923, 2010.

[39] B. Fernandez-Gauna, J.M. Lopez-Guede, and M. Graña. Transfer learning with partially constrained models: application to reinforcement learning of linked multicomponent robot system control. *Robotics and Autonomous Systems*, submited, 2012.

[40] B. Fernandez-Gauna, J.M. Lopez-Guede, and E. Zulueta. Linked multicomponent robotic systems: Basic assessment of linking element dynamical effect. In E. Corchado, M. Graña, and A. Savio, editors, *Hybrid Artificial Intelligence Systems, Part I*, volume 6076, pages 73–79. Springer Verlag, 2010.

[41] Borja Fernandez-Gauna, Jose M. Lopez-Guede, Ekaitz Zulueta, Zelmar Echegoyen, and Manuel 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.

[42] Zelmar Echegoyen Ferreira. *Contributions to Visual Servoing for Legged and Linked Multicomponent Robots*. PhD thesis, UPV/EHU, 2009.

[43] 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.

[44] Peter Geibel. Reinforcement learning with bounded risk. In *In Proceedings of the Eighteenth International Conference on Machine Learning*, pages 162–169. Morgan Kaufmann, 2001.

[45] Peter Geibel. Reinforcement learning for mdps with constraints. In *ECML*, pages 646–653, 2006.

[46] Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *J. Artif. Int. Res.*, 24:81–108, July 2005.

[47] 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.

[48] Mohammad Ghavamzadeh, Sridhar Mahadevan, and Rajbala Makar. Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13:197–229, September 2006.

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

[50] M. Gregoire and E. Schomer. Interactive simulation of one-dimensional flexible parts. *Computer-Aided Design*, 39(8):694–707, 2007.

[51] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia. Generalizing plans to new environments in relational MDPs. In *In International Joint Conference on Artificial Intelligence (IJCAI-03*, pages 1003–1010, 2003.

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

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

[54] 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.

[55] Alexander Hans, Daniel Schneegaß, Anton Maximilian Schäfer, and Steffen Udluft. Safe exploration for reinforcement learning. In *ESANN*, pages 143–148, 2008.

[56] M. Heger. Consideration of risk in reinforcement learning. In *XI. International Machine Learning Conference*, 1994.

[57] 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.

[58] Bernhard Hengst. Model approximation for hexq hierarchical reinforcement learning. In *In ECML 2004*, pages pp. 144–155, 2004.

[59] E. Hergenröther and P. Dhne. Real-time virtual cables based on kinematic simulation. In *Proocedings of the WSCG*, 2000.

[60] Pieter Hoen, Karl Tuyls, Liviu Panait, Sean Luke, and J. A. La Poutré. 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.

[61] 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.

[62] 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.

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

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

[65] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[66] 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.

[67] Josef Kittler, Mohamad Hatef, Robert P. W. Duin, and Jiri Matas. On combining classifiers. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 20:226–239, 1998.

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

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

[70] 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.

[71] Ludmila Ilieva Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, 2004.

[72] 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.

[73] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. A.: Transfer of samples in batch reinforcement learning. In *In: Proceedings of the 25th Annual ICML*, pages 544–551, 2008.

[74] Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for mdps. In *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages pp. 531–539, 2006.

[75] 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.

[76] 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.

[77] 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.

[78] Carlos E. Mariano and Eduardo Morales. 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*, pages 12–15, 2000.

[79] Geoffrey Holmes Bernhard Pfahringer Peter Reutemann Ian H. Witten Mark Hall, Eibe Frank. The weka data mining software: An update. In *SIGKDD Explorations*, volume 11, 2009.

[80] L. Matignon, G.J. Laurent, and N. Le Fort-Piat. 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*, pages 64 –69, 29 2007-nov. 2 2007.

[81] 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.

[82] T.W. McLain and R.W. Beard. Coordination variables, coordination functions, and cooperative timing missions. *AIAA Journal of Guidance, Control, & Dynamics*, 28(1):150–161, 2005.

[83] 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.

[84] 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.

[85] 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.

[86] H.E. Osman and H. Osamu. Online incremental random forests. In *Machine Vision, 2007. ICMV 2007. International Conference on*, pages 102 –106, dec. 2007.

[87] D.K. Pai. Strands: Interactive simulation of thin solids using cosserat models. *Computer Graphics Forum*, 21(3):347–352, 2002.

[88] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:387–434, 2005. 10.1007/s10458-005-2631-2.

[89] 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.

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

[91] 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.

[92] H. Qin and D. Terzopoulos. D-nurbs: A physics-based framework for geomatric design. Technical report, Los Alamitos, CA. USA, 1996.

[93] Jan Ramon, Kurt Driessens, and Tom Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *In Proc. of The 18th European Conf. on Machine Learning*. Springer-Verlag, 2007.

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

[95] 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.

[96] M.B. Rubin. *Cosserat Theories: Shells, Rods and Points*. Kluwer, 2000.

[97] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. On-line random forests. In *3rd IEEE ICCV Workshop on On-line Learning for Computer Vision*, 2009.

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

[99] Anton Maximilian Schäfer and Steffen Udluft. Solving partially observable reinforcement learning problems with recurrent neural networks. In *In Workshop Proc. of the European Conference on Machine Learning*, 2005.

[100] 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.

[101] Oliver G. Selfridge, Richard S. Sutton, and Andrew G. Barto. Training and tracking in robotics. In *Proceedings of the 9th international joint conference on Artificial intelligence - Volume 1*, pages 670–672, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc.

[102] 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.

[103] 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.

[104] 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.

[105] 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.

[106] Richard Sutton and Andrew G. G. Barto. *Reinforcement Learning I: Introduction*. MIT Press, 1998.

[107] 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.

[108] 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.

[109] 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.

[110] 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.

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

[112] 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.

[113] Matthew E. Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *In Proceedings of the Twenty-Fourth International Conference on Machine Learning*, pages 879–886, 2007.

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

[115] Matthew E. Taylor, Peter Stonedepartment, and Computer Sciences. Representation transfer for reinforcement learning. In *In AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development*, 2007.

[116] A. Theetten, L. Grisoni, C. Andriot, and B. Barsky. Geometrically exact dynamic splines. *Computer-Aided Design*, 40(1):35–48, January 2008.

[117] Peter Vamplew, Richard Dazeley, Adam Berry, Rustam Issabekov, and Evan Dekker. Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 1:1–30, 2011.

[118] 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.

[119] Aiping Wang, Guowei Wan, Zhiquan Cheng, and Sikun Li. An incremental extremely random forest classifier for online learning and tracking. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 1449 –1452, nov. 2009.

[120] 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.

[121] C. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, Psychology department, 1989.

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

[123] 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.

[124] Hairong Xiao, Li Liao, and Fengyu Zhou. Mobile robot path planning based on q-ann. In *Automation and Logistics, 2007 IEEE International Conference on*, pages 2650 –2654, 2007.

[125] 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.