UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

FELIPE LENO DA SILVA

METHODS AND ALGORITHMS FOR KNOWLEDGE REUSE IN MULTIAGENT REINFORCEMENT LEARNING

São Paulo
2019

FELIPE LENO DA SILVA

METHODS AND ALGORITHMS FOR KNOWLEDGE REUSE IN MULTIAGENT REINFORCEMENT LEARNING

Thesis submitted to Escola Politécnica da Universidade de São Paulo in fulfillment of the requirements for the degree of Doctor of Science

São Paulo
2019

FELIPE LENO DA SILVA

METHODS AND ALGORITHMS FOR KNOWLEDGE REUSE IN MULTIAGENT REINFORCEMENT LEARNING

Thesis submitted to Escola Politécnica da Universidade de São Paulo in fulfillment of the requirements for the degree of Doctor of Science

Research Area:

Computer Engineering

Advisor:

Prof. Dr. Anna Helena Reali Costa

São Paulo
2019

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Silva, Felipe Leno da
Methods and algorithms for knowledge reuse in multiagent reinforcement learning / F. L. Silva, A. H. R. Costa -- versão corr. -- São Paulo, 2019.
130 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Inteligência Artificial 2.Aprendizagem de máquina 3.Aprendizagem por reforço 4.Sistemas multiagentes I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t. III.Costa, Anna Helena Reali

Who am I, and what is my family, that you have brought me this far?

Adapted from 2 Samuel 7:18, the Holy Bible.

Acknowledgments

A Doctorate is a very long and harsh path that I could never say I have walked alone. In the past years, I have come across many people that collaborated in their own way so that this moment would come.

Firstly I thank my parents, Berto and Solange, without whom getting here would never be possible. When financial resources were not abundant, they taught me the importance of education and incentivized me to never settle down and always strive to improve myself. After all, this was all that was needed.

To my significant one, Juliana, for the many years during which she had to divide my time with my work, and the Ph.D. took a too-big portion of it.

Many thanks to my sister, Paula, for all the moments we have shared during and before the course of my Ph.D. (and for the occasional reviews of texts and presentations).

To my advisor, Anna Helena Reali Costa, for the guidance during all those years, and for teaching me so much about every aspect of being a Researcher.

To all members of the Intelligent Techniques Laboratory (LTI) at the University of São Paulo, where I probably spent more time than in my own house in the last years. Special thanks to (roughly in the order they joined the group): Walter Mayor Toro, Juan Perafan Villota, Heider Berlink, Ricardo de Souza Jacomini, Allan Lima, Ruben Glatt, Vinicius de Carvalho, Rodrigo Cesar Bonini, and Amir Saad.

My gratitude to Peter Stone, for receiving me at UT Austin and advising me as one of his own students for one year.

My appreciation to all members of the Learning Agents Research Group (LARG)

at UT Austin for receiving me so well. My special thanks to Josiah Hanna, Patrick MacAlpine, Elad Liebman, Ruohan Zhang, Sanmit Narvekar, Harel Yedidsion, Garrett Warnell, Katie Genter, Ishan Durugkar, Eddy Hudson, Justin Hart, Harsh Goyal, and Faraz Torabi.

To Matthew E. Taylor for inviting me for a very productive internship, and to my main collaborators at Borealis AI Pablo Hernandez-Leal and Bilal Kartal.

I also thank Ellen Elmore and Mister Manini for the good time we had in Austin, when I was literally thousands of miles from all that I knew.

To Edith Ranzini, Fernando Giorno, and Francisco Marcondes, for incentivizing me to apply for the master's course right after I finished the undergrad, which started all the good things that happened in the last years.

To my Elementary School teachers, especially "Prô" Angela Ferro, for showing me that dedication and hard work prevail even in the worst of conditions.

Finally, I gratefully acknowledge the funding agencies that made this research possible: São Paulo Research Foundation (FAPESP), grants 2015/16310-4, 2016/21047-3, and 2018/00344-5; *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES), Finance Code 001; and *Fundação para o Desenvolvimento Tecnológico da Engenharia* (FDTE).

Epigraph

Somewhere, something incredible
is waiting to be known.

Carl Sagan

Study while others are sleeping;
work while others are loafing;
prepare while others are playing;
and dream while others are wishing.

William A. Ward

Resumo

O Aprendizado por Reforço (*Reinforcement Learning* - RL) é uma das técnicas mais bem-sucedidas para treinar agentes através de interações com o ambiente. Entretanto, o processo de aprendizado tem uma alta complexidade em termos de amostras de interação com o ambiente para que uma política efetiva seja aprendida, especialmente quando múltiplos agentes estão atuando simultaneamente. Este trabalho propõe reusar conhecimento prévio para acelerar o aprendizado em RL multiagente. Os agentes podem reusar conhecimento adquirido em tarefas resolvidas previamente, e também podem receber instruções de agentes com mais experiência para aprender mais rápido. Porém, especificar um arcabouço que integre reuso de conhecimento no processo de aprendizado requer responder questões de pesquisa desafiadoras, tais como: Como abstrair soluções para que sejam reutilizadas no futuro em tarefas similares porém diferentes? Como definir quando aconselhamentos entre agentes devem ocorrer? Como selecionar as tarefas passadas mais similares à nova a ser resolvida e mapear correspondências? e Como definir se um conselho recebido é confiável?

Apesar de diversos métodos existirem para o reúso de conhecimento de uma fonte em específico, a literatura é composta por métodos especializados em um determinado cenário, que não são compatíveis com outros métodos. Nesta tese é proposto o reúso de conhecimento tanto de tarefas prévias como de outros agentes. Para cumprir este objetivo, diversos métodos flexíveis são propostos para que cada um destes dois tipos de reúso de conhecimento seja possível. Os métodos propostos incluem: *Ad Hoc Advising*, no qual agentes compartilham conhecimento através de sugestões de ações; e uma extensão da representação orientada a objetos para RL multiagente e métodos para aproveitá-la no reúso de conhecimento. Combinados, os métodos propostos propõem formas de se reusar o conhecimento proveniente tanto de tarefas prévias quanto de outros agentes com desempenho do estado da arte. As contribuições dessa tese são passos iniciais na direção a métodos mais flexíveis de transferência de conhecimento multiagentes, onde agentes serão capazes de combinar consistentimente conhecimento reusado de múltiplas origens, incluindo tarefas resolvidas e outros agentes.

Palavras-chave: Aprendizado por Reforço Multiagente, Transferência de Conhecimento, Aprendizado por Reforço, Sistemas Multiagente

Abstract

Reinforcement Learning (RL) is a well-known technique to train autonomous agents through interactions with the environment. However, the learning process has a high sample-complexity to infer an effective policy, especially when multiple agents are simultaneously actuating in the environment. We here propose to take advantage of previous knowledge, so as to accelerate learning in multiagent RL problems. Agents may reuse knowledge gathered from previously solved tasks, and they may also receive guidance from more experienced friendly agents to learn faster. However, specifying a framework to integrate knowledge reuse into the learning process requires answering challenging research questions, such as: How to abstract task solutions to reuse them later in similar yet different tasks? How to define when advice should be given? How to select the previous task most similar to the new one and map correspondences? and How to defined if received advice is trustworthy? Although many methods exist to reuse knowledge from a specific knowledge source, the literature is composed of methods very specialized in their own scenario that are not compatible. We propose in this thesis to reuse knowledge both from previously solved tasks and from communication with other agents. In order to accomplish our goal, we propose several flexible methods to enable each of those two types of knowledge reuse. Our proposed methods include: *Ad Hoc Advising*, an inter-agent advising framework, where agents can share knowledge among themselves through action suggestions; and an extension of the *object-oriented* representation to multiagent RL and methods to leverage it for reusing knowledge. Combined, our methods provide ways to reuse knowledge from both previously solved tasks and other agents with state-of-the-art performance. Our contributions are first steps towards more flexible and broadly applicable multiagent transfer learning methods, where agents will be able to consistently combine reused knowledge from multiple sources, including solved tasks and other learning agents.

Keywords: Multiagent Reinforcement Learning, Transfer Learning, Reinforcement Learning, Multiagent Systems

List of Figures

1.1 High-level general view of the area of Transfer in Multiagent RL. . . . 2

2.1 The *Goldmine* domain. *Miners* aim to gather all *gold pieces* in the environment. Thick *walls* are impassable (adapted from (DIUK, 2009)). (a) Graphical representation. (b) Object representation of the state drawn in (a). . . . 9

2.2 Relation between concepts in an OO-MDP (adapted from (DIUK, 2009)). 11

2.3 Illustration of transfer performance metrics. . . . 15

2.4 Illustrations of different settings of the *Gridworld* domain. From top-left to bottom-right: *Predator-Prey* (SILVA; COSTA, 2017c), *Goldmine* (DIUK, 2009), *Multi-room Gridworld* (KOLTER; ABBEEL; NG, 2008), *Dog Training* (PENG et al., 2016a), *Gridworld with beacons* (NARVEKAR; SINAPOV; STONE, 2017), *Multiagent Gridworld Navigation* (HU; GAO; AN, 2015b), *Theseus and the Minotaur* (MADDEN; HOWLEY, 2004), *Taxi* (DIETTERICH, 2000). . . . 16

2.5 Illustration of the Predator-Prey domain. The red square depicts the visual field (depth=3) of the green predator. . . . 17

2.6 Screenshot of the Pong domain. . . . 18

2.7 The HFO environment with three offensive agents against one goalkeeper. . . . 19

3.1 Illustration of our classification of the current literature on TL for multiagent RL. Groups to the left are more general and contain the groups to the right. . . . 24

4.1 The average number of steps to score a goal (top-left), average spent budget (top-right), and percentage of goals (bottom) observed in the evaluation steps for each algorithm learning from scratch. The numbers are averages from 50 runs and the shaded area corresponds to the 95% confidence interval. The performance achieved by a random agent was included as baseline. . . . 51

4.2 The average percentage of goals observed in the evaluation steps for the *Mistake Correcting Advising*. The numbers are averages from 50 runs and the shaded area corresponds to the 95% confidence interval. The Torrey performance without sharing the intended action was included as a baseline. 52

4.3 The average spent budget per evaluation steps for each algorithm in the experiments of this section. 52

4.4 The average percentage of goals observed in the evaluation steps for each algorithm when one agent is an expert. The numbers are averages from 50 runs and the shaded area corresponds to the 95% confidence interval. The NoAdvice performance when all agents are learning from scratch was included as a baseline. 53

4.5 (a) Illustration of a regular DQN network and (b) a network with *heads*. Each head estimates a value for each action. 57

4.6 Average uncertainty of the learning episodes over time. Averaged over 200 repetitions. 60

4.7 (a) Discounted rewards and (b) amount of advice used in 200 repetitions of the *Gridworld* experiment. The shaded area corresponds to the 90% confidence interval. The dashed line corresponds to the optimal performance. 60

4.8 Sum of discounted rewards observed in 200 repetitions of the *Gridworld* experiment. The shaded area corresponds to the 90% confidence interval. 61

4.9 (a) Undiscounted rewards and (b) amount of advice used in 20 repetitions of the *Pong* experiment. The shaded area corresponds to the 60% confidence interval. 62

4.10 Sum of undiscounted rewards observed in 20 repetitions of the *Pong* experiment. 62

5.1 (a) Graphical representation and (b) textual representation of the state space abstraction. s_1 and s_2 represent two concrete states (ids on top of gold pieces) that are described by a single abstract state $\tilde{s}$ in the right side when $Gold \in \Gamma$. 65

5.2 Observed discounted cumulative reward in the *Goldmine* domain. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The vertical axis represents the metric evaluated every 100 episodes of exploration. The shaded area represents the 95% confidence interval observed in 70 repetitions. 72

5.3 Observed number of steps to complete one episode in the *Goldmine* domain. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The vertical axis represents the metric evaluated every 100 episodes of exploration. The shaded area represents the 95% confidence interval observed in 70 repetitions. 72

5.4 Observed discounted cumulative reward in the *Gridworld* domain. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The vertical axis represents the distributed accumulated reward evaluated every two episodes of exploration. The shaded area represents the 95% confidence interval observed in 100 repetitions. 74

5.5 Observed average number of steps to capture the prey in evaluation episodes. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The shaded area represents the 99% confidence interval observed in 1000 repetitions. The performance achieved by a random agent is included as a baseline. 75

5.6 Observed Q-table size in the *Predator-Prey* domain. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The vertical axis represents the average Q-table size at that step. 75

5.7 Amplification of Figure 5.6 to depict the difference between *DQL* and *DOO-Q*. 76

5.8 Predator movement after using actions *North* (a), *South* (b), *West* (c), and *East* (d) in Experiment 3. 82

5.9 The average of steps to solve the task during evaluation episodes in 100 repetitions for: (a) *Predator-Prey* and (b) *Goldmine* domains. The shaded area corresponds to the 99% confidence interval. The performance of a random agent is included as baseline. 83

5.10 The average performance 50 repetitions of Experiment 1. The shaded area corresponds to the 99% confidence interval. The performance of a random agent is included as baseline. 83

5.11 The average performance 50 repetitions of Experiment 2. The shaded area corresponds to the 99% confidence interval. The performance of regular learning in Experiment 1 was included to allow comparisons in regard to reward function effectiveness. 84

5.12 The average performance 75 repetitions of Experiment 3. The shaded area corresponds to the 99% confidence interval. 85

5.13 The average performance 50 repetitions of Experiment 4. The shaded area corresponds to the 99% confidence interval. 85

5.14 An illustration of the target task in Gridworld. 94

5.15 The average discounted rewards observed in 2,000 repetitions of the experiment in the *Gridworld* domain. (a) refers to the average performance during learning; and (b) to the cumulative rewards starting from step 1,500. Steps used to learn source tasks are also considered in the graph. The shaded area is the 95% confidence interval. 96

5.16 The average discounted rewards disregarding steps carried out in source tasks in the *Gridworld* domain. The shaded area is the 95% confidence interval. 97

5.17 The average percentage of goals observed in 50 repetitions of the experiment in the *HFO* domain when the steps used to learn source tasks are: (a) considered; and (b) not considered. The shaded area is the 95% confidence interval. 98

7.1 Fictitious performance comparison of two algorithms. (a) Running 10 episodes; (b) Running 20 episodes. 104

7.2 Illustration of how a web-based knowledge base could be built. The agent accesses an ontology base for defining abstract state and action spaces. Then, an online knowledge base provides knowledge to the agent. Inspired by Kono *et al.*'s illustration (KONO et al., 2014) . . . 108

List of Tables

3.1 Quick-Reference legend for abbreviations used in Tables 3.3, 3.4, and 3.5. 30

3.2 Description of our contributions by their categories. Symbols follow the convention of Table 3.1. 30

3.3 Summary of main recent trends of *Intra-Agent Transfer* methods. We follow the symbols introduced in Section 3.1 and compiled in Table 3.1 for quick reference. The publications are presented in chronological order within their group. 32

3.4 Summary of main recent trends of *Inter-Agent Transfer* methods. We follow the symbols introduced in Section 3.1 and compiled in Table 3.1 for quick reference. Publications are presented in chronological order within their group. For all papers **AD**= $\mathcal{S}$. 36

3.5 Summary of main recent trends of *Inter-Agent Transfer* methods. Second part of Table 3.4. 37

5.1 Manual Inter-Task Mapping translating the target task with 2 preys and 4 predators to a source task with 1 prey and 3 predators. Here, z is the local agent, *Prey* is the set of prey objects, *Predator* is the set of predator objects, and $dist$ is an Euclidian distance function. 81

5.2 Summary of all experimental results. The value here represented are the average number of steps to solve the task in each experiment, where $jumpstart$ is the performance after 0 learning episodes and $asymptotic$ is the performance after 4500 episodes.Best results (according to statistical significance tasks) are denoted in bold. 86

List of Abbreviations

RL	Reinforcement Learning
MAS	Multiagent System
TL	Transfer Learning
MDP	Markov Decision Process
SG	Stochastic Game
OO-MDP	Object-Oriented Markov Decision Process
DRL	Deep Reinforcement Learning
DQN	Deep Q-Network
KL	Kullback–Leibler (divergence)
HFO	Half Field Offense
AdHocTD	Ad Hoc Temporal Difference advising
AdHocVisit	Ad Hoc Visit-Based advising
RCMP	Requesting Confidence-Moderated Policy advice
A3C	Asynchronous Advantage Actor Critic
MOO-MDP	Multiagent Object-Oriented MDP
DOO-Q	Distributed Object-Oriented Q-Learning
SAQL	Single-agent Q-Learning
MAQ	Multiagent Q-Learning
DQL	Distributed Q-Learning
PITAM	Probabilistic Inter-TAsk Mapping

List of Symbols

S	Set of states
A	Set of actions
T	State transition function
R	Reward Function
γ	Discount factor
π	A policy
Q	Q-table function
s_k	Subscripts refer to time steps unless otherwise indicated, e.g., s_k is the current state at k
Q^*	The symbol * refers to optimal functions, e.g., optimal Q-function Q^*
α	Learning rate
U	Joint action set
R^i	Superscripts refer to the origin of an element unless otherwise indicated, e.g., R^i is the reward function that belongs to agent i
Φ	An Equilibrium value
C	A set of classes
C_i	A class
$Att(C_i)$	The set of attributes that compose class C_i
$Dom(att)$	Domain of attribute *att*
ϱ	A set of *terms*
D	A set of *rules*
$\mathcal{L}$	A loss function
θ	A set of parameters
F_θ	A function F parameterized by θ
$\mathscr{K}$	The available knowledge to solve a task
$\mathscr{H}$	The set of policies that an RL algorithm can learn
$\mathscr{A}$	An RL algorithm
$\boldsymbol{u_k}$	A *joint* action

G	A set of Agents
P_{ask}	Function to define the probability of asking for advice
P_{give}	Function to define the probability to give advice
b_{ask}	Budget to receive advice
b_{give}	Budget to give advice
Ω	Function to combine multiple advice
ζ	State translation function
Υ	Policy confidence function to advisees
Ψ	Policy confidence function to advisors
π_{Δ}	Demonstrator policy
μ	Uncertainty estimator
$\boldsymbol{A}_t$	Availability function
Ag	Set of Agent classes
Γ	Set of *abstracted* classes
$o.\overline{state}$	The abstract state of o
E	A Set of environment objects
$\tilde{s}$	An abstract state
κ	State abstraction function
$\mathscr{X}_C$	A Class Mapping
$\mathscr{P}_o$	A Probabilistic Inter-Task Mapping
Θ	Set of mapped states
$\mathscr{X}_a$	An Action Mapping
ω	An object-oriented mapping
$\mathcal{T}$	Set of tasks
$\mathcal{T}_s$	A task
$\mathbb{C}$	A *Curriculum*
ν	A transfer potential function
$\boldsymbol{\mathcal{V}}$	Vertexes in a Curriculum graph
$\boldsymbol{\mathcal{E}}$	Edges in a Curriculum graph

Contents

1 Introduction **1**

1.1 Objective 3

1.2 Contributions 3

2 Background **5**

2.1 Reinforcement Learning 5

2.2 Multiagent RL 7

2.3 Object-Oriented MDP 9

2.4 Deep RL 12

2.5 Curriculum Learning for RL 13

2.6 Transfer Learning 14

2.7 Evaluation Domains 15

2.7.1 Gridworld 16

2.7.2 Predator-Prey 17

2.7.3 Goldmine 18

2.7.4 Pong 18

2.7.5 HFO 18

3 Proposed Transfer Learning Taxonomy and Related Work **21**

3.1 Proposed Taxonomy 22

3.1.1 Nomenclatures 24
3.1.2 Learning Algorithm (LA) 25
3.1.3 Source Task Selection (ST) 26
3.1.4 Mapping Autonomy (MA) 27
3.1.5 Transferred Knowledge (TK) 27
3.1.6 Allowed Differences (AD) 29
3.2 Methods Proposed in this thesis 30
3.3 Intra-Agent Transfer Methods 31
3.3.1 Adapting to Other Agents 31
3.3.2 Sparse Interactions Algorithms 31
3.3.3 Relational Descriptions 33
3.3.4 Source Task Selection 33
3.3.5 Curriculum Learning 33
3.3.6 Biases and Heuristics 35
3.4 Inter-Agent Transfer Methods 35
3.4.1 Action Advising 35
3.4.2 Human-focused Transfer 38
3.4.3 Reward Shaping and Heuristics 38
3.4.4 Learning from Demonstrations 39
3.4.5 Imitation 39
3.4.6 Curriculum Learning 40
3.4.7 Inverse Reinforcement Learning 40
3.4.8 Transfer in Deep Reinforcement Learning 40
3.4.9 Scaling Learning to Complex Problems 41

4 Simultaneously Learning and Advising **43**
4.1 Ad Hoc Advising 43
4.1.1 Advice Strategy for Simultaneously Learning Agents 44

4.1.2 Deciding When to Ask for and When to Give Advice 46
4.1.3 Experimental Evaluation 48
4.1.4 Discussion 54
4.2 Scaling up Ad Hoc Advising 54
4.2.1 Uncertainty-Aware Advice for Deep RL 55
4.2.2 Implementation-friendly Description 57
4.2.3 Experimental Evaluation 58
4.2.4 Discussion 62

5 Object-Oriented Representation for Transfer 63
5.1 Object-Oriented Representation in MAS 63
5.1.1 Learning in deterministic cooperative MOO-MDPs 66
5.1.2 Experimental Evaluation 68
5.1.3 Goldmine Results 70
5.1.4 Gridworld Results 73
5.1.5 Predator-Prey Results 74
5.2 Inter-Task Mapping 76
5.2.1 Transferring Knowledge using PITAM 78
5.2.2 Experimental Evaluation 80
5.2.3 Results 82
5.2.4 Discussion 86
5.3 Curriculum Generation 86
5.3.1 Object-Oriented Curriculum Generation 87
5.3.2 Automatic Source Task Generation 91
5.3.3 Experimental Evaluation 93
5.3.4 Results Gridworld 95
5.3.5 Results HFO 97
5.3.6 Discussion 98

6 Conclusions **101**

7 Further Work **103**

REFERENCES **111**

A List of Publications **125**

B Proof for Equation (5.2) **127**

C Proof for Learning a Greedy Joint Policy with DOO-Q **129**

Chapter 1

Introduction

Building intelligent agents to effectively and autonomously solve tasks is one of the primary goals of Artificial Intelligence researchers.

A widely used and flexible solution to develop and train autonomous agents is to bestow upon them the ability to learn from experimentation, through direct actuation in the environment. This solution is known as *Reinforcement Learning* (RL) (LITTMAN, 2015; SUTTON; BARTO, 1998), and consists of choosing actions to apply in the environment to observe how good each action is under each situation.

RL has been extensively used to solve sequential decision-making problems in a wide range of increasingly complex domains (MNIH et al., 2015; NG et al., 2006; STONE; SUTTON; KUHLMANN, 2005; TESAURO, 1995). However, the classical RL approach is not scalable and requires a huge number of samples of interactions with the environment to learn how to solve a task, which is unfeasible for many domains where applying wrong actions may be very harmful or costly.

Scalability issues are further intensified in Multiagent domains, as the decision maker may need to reason over the other agents in the environment. In a world where more and more devices have computing power, many difficult tasks can (sometimes must) be solved by *Multiagent Systems* (MAS). Modeling problems as MAS enables the construction of systems that solve subtasks in paralel and are robust to individual device failures (WOOLDRIDGE, 2009). Therefore, scalable and robust techniques are of utmost importance for Multiagent RL algorithms.

Many different lines of work proposed techniques to alleviate the sample complexity of RL and accelerate learning. *Transfer Learning* (TL) (TAYLOR; STONE, 2009; PAN; YANG, 2010) proposes to reuse previously acquired knowledge similarly as humans do. For example, learning Spanish beforehand knowing Portuguese is much

easier than trying to learn it with no knowledge of any similar language. Similarly, a robot learning how to play soccer may reuse knowledge from a previous task in which it learned how to walk while carrying a ball. Furthermore, a more experienced teammate could advice the agent on when to shoot to maximize the goal chance, or the agent could observe its teammates' behavior to imitate strategies and moves.

Specifically applied to RL, TL has been successfully used in many ways, accelerating the learning process in both single- and multiagent domains (TAYLOR; STONE; LIU, 2007; TAYLOR et al., 2014a; ZHAN; BOU-AMMAR; TAYLOR, 2016; KOGA; SILVA; COSTA, 2015). However, in spite of the encouraging results achieved on the last years, there are no consensus on many aspects that must be defined to specify a suitable TL algorithm.

Figure 1.1 depicts a general view of how the area is currently divided. The agent extracts knowledge from advice given by other agents or from previously solved tasks to accelerate learning of a new task. The solution of new tasks can be abstracted, so that reusing it for similar yet different tasks later will be easier, and added to the knowledge base. In the long-term, the agent is expected to learn tasks much faster due to the task solutions stored in its knowledge base and the received advice, which is usually specific for the current task.

In order to make RL applicable in complex multiagent domains, we would expect agents to be able to combine knowledge from both sources (i.e. previous tasks and other agents) consistently. However, most of the literature aims at improving or proposing methods to solve narrowly one of those problems (or subproblems). Also, most of those methods are not inter-compatible, which means that the methods cannot be easily combined in an application.

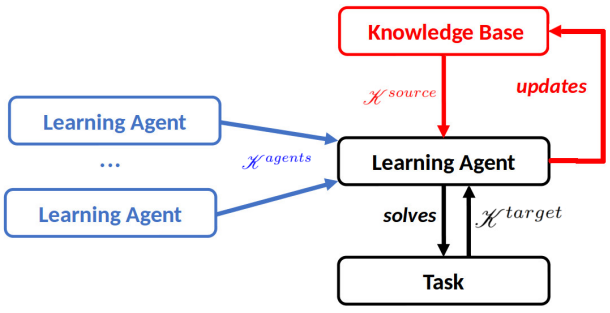


Figure 1.1. High-level general view of the area of Transfer in Multiagent RL.

Therefore, our aim in this thesis is to propose flexible **Transfer Learning methods to accelerate Multiagent Reinforcement Learning** through knowledge reuse, with the intention of bringing the methods closer to the aforementioned scenarios and enabling their integration in the long term. Combining such methods and applying

them to the learning process requires answering the following challenging questions, that have been only partially answered by previous works: (i) *How to abstract acquired knowledge, in order to better generalize and reuse it?* (ii): *How to represent knowledge?* (iii): *When and how to transmit knowledge efficiently?* and (iv): *How to evaluate the quality of transferred knowledge?*

1.1 Objective

Our objective in this research is to specify flexible *Transfer Learning* methods to autonomously and consistently accelerate learning in *Multiagent Reinforcement Learning Systems*, both by reusing knowledge from previously solved tasks and communication with other (possibly more experienced) agents.

1.2 Contributions

In this section, we summarize the main contributions of this thesis, as well as in which chapter their description can be found.

The main purpose of the methods proposed throughout this thesis is to leverage both knowledge from other agents ($\mathscr{K}^{agents}$) and knowledge from previous tasks ($\mathscr{K}^{source}$) to accelerate learning. The current literature is composed of methods with little inter-compatibility. Although $\mathscr{K}^{agents}$ and $\mathscr{K}^{source}$ have been reused individually, most of state-of-the-art methods are built for very specific scenarios and there is no easy way to combine multiple of them. We here contribute flexible methods that could be simultaneously employed in a multiagent system, enabling the reuse of both knowledge from previous tasks and other agents at the same time. According to the needs and characteristics of the MAS to be developed, one or multiple of our proposed methods can be employed. We also show empirically that all methods achieve state-of-the-art performance when evaluated individually in their category of TL.

Our first contribution is a taxonomy of Transfer for Multiagent RL and a survey of state-of-the-art techniques (Chapter 3). We provide a novel categorization of the area and discuss all related state-of-the-art works put in the perspective of the challenges we are tackling in this thesis.

The *Ad Hoc Advising* method fully described in Chapter 4 is useful for reusing $\mathscr{K}^{agents}$, where agents exchange knowledge through action advising. The general idea of the method is to leverage a confidence function to identify portions of the state-

action state where an agent is not confident and ask for advice to other agents only in those situations. The method is especially efficient if some of the agents have more knowledge than the others for some situations, either because they have been training for a longer time or because they have reused previous knowledge (e.g., using one of the other methods proposed throughout this thesis). In this method we were able to achieve performance improvement while removing key restrictions in most of the related literature: (a) there is no restriction on the learning algorithm of agents involved on advising relations; (b) constant supervision of the learning agent is not required; and (c) none of the agents in the system is required to have a high-performance since the beginning of the learning process. Those restrictions were the main limiting factor for combining this method with other types of TL.

We also propose a group of methods in Chapter 5 that leverage the object-oriented representation to reuse $\mathscr{K}^{source}$. We take advantage of the relational description to facilitate knowledge abstraction and to enable TL. First, we contribute by proposing an object-oriented description to multi-agent systems and then propose a learning algorithm based on this new description. We also use it to map correspondences between different tasks and thus provide a more automatic mode for learning transfer. Finally, we show that this same representation can be used to decompose a complex task into simpler ones and sort them into a sequence of tasks to be followed during learning. Related methods usually are based on less-intuitive relational representations and perform transfer only through very similar domains. Moreover, similar approaches usually use manually defined sets of source tasks. Our methods provide an intuitive way of describing the tasks that is useful to perform transfer, as well as to automatically decomposed complex tasks and order them in an appropriate sequence. Those methods are easily applicable in a wide range of scenarios, and could be easily combined with *Ad Hoc Advising*.

The flexible state-of-the-art methods proposed in this thesis spam across both reusing $\mathscr{K}^{source}$ and $\mathscr{K}^{agents}$. This thesis is the first work that explicitly considers the development of methods reusing both types of knowledge transfer.

Chapter 2

Background

This chapter describes all basic concepts needed to fully understand this thesis, as well as the domains used in our experiments.

2.1 Reinforcement Learning

RL is a solution for Markov Decision Processes (MDP) (PUTERMAN, 2005), which is a popular model for sequential decision-making problems.

An MDP is a tuple $\langle S, A, T, R, \gamma \rangle$, where:

- S is the (possibly infinite) set of environment states. A common and convenient way to describe states is using a factored description, defining state variables and building states according to their values. An initial state distribution function $S_0 : S \rightarrow [0, 1]$ defines the probability of starting in each state when the task restarts (i.e., a new episode begins);
- A is the set of available actions to the agent;
- $T : S \times A \times S \rightarrow [0, 1]$ is a stochastic state transition function, where the state is transitioned to state s' with a probability $0 \leq p \leq 1$ when applying action a in state s. We denote as $s' \leftarrow T(s, a)$ drawing a sample of next state from T;
- $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function; and
- $\gamma \in [0, 1)$, is the discount factor, which represents the relative importance of future and present rewards.

At each step, the reasoning agent observes the current state $s \in S$. Then, it chooses an action a among the applicable ones in s, causing a state transition $s' \leftarrow T(s, a)$. After each action, the agent receives a reward signal $r \leftarrow R(s, a, s')$, that represents the quality of the executed action towards the task solution. In learning problems, the functions T and R are unknown to the agent, hence a proper actuation must be induced through the observation of $\langle s, a, s', r \rangle$ tuples, gathered through interactions with the environment. The goal of the learning agent is to induce a policy $\pi : S \rightarrow A$, that maps an action to be applied in each possible state.

A possible way to learn a good policy is by iteratively updating an estimate of action qualities $Q : S \times A \rightarrow \mathbb{R}$ after each interaction with the environment. Several algorithms (e.g., Q-Learning) are proved to eventually learn the true Q function under non-restrictive assumptions[1]:

$$Q^*(s, a) = max_\pi \mathop{\mathbb{E}}_{s,a} \left[\sum_{k=0}^{\infty} \gamma^k r_k | \pi \right], \tag{2.1}$$

where r_k is the reward received after k steps from using action a in state s and following the optimal policy on all subsequent steps, and γ is a discount factor. Therefore, an optimal policy π^* is one that maximizes the expected sum of discounted rewards in every possible state. For the Q-Learning algorithm, Q is updated after each applied action by:

$$Q_{k+1}(s_k, a_k) \leftarrow (1 - \alpha) Q_k(s_k, a_k) + \alpha (r_k + \gamma \max_a Q_k(s_{k+1}, a)), \tag{2.2}$$

where $r_k = R(s_k, a_k, s_{k+1})$, $0 < \alpha \leq 1$ is the learning rate and γ is the discount factor.

After learning Q^*, the agent can use it to define an optimal policy:

$$\pi^*(s) = argmax_a Q^*(s, a). \tag{2.3}$$

Notice that MDPs take into account only one agent in the environment. Although other agents could be ignored to learn in a MAS as if it was a single-agent problem (TAN, 1993), the actions of one agent often have influence in the local state and/or reward of the others. Thus, for most cases, coordination is desired.

[1] The full proof for Q-Learning is available at (WATKINS; DAYAN, 1992). The main conditions are that: (i) all state-action pairs are infinitely visited; (ii) the rewards are bounded; (iii) a proper learning rate is chosen.

2.2 Multiagent RL

MDPs are extended to MAS as Stochastic Games (SG) (LITTMAN, 1994; BUSONIU; BABUSKA; SCHUTTER, 2008; BOWLING; VELOSO, 2000). As more than one agent is now present in the environment, an SG is composed of $\langle S, U, T, R^{1\ldots n}, \gamma \rangle$, where:

- n is the number of agents;
- S is the state space. Each state is composed of local states from each agent plus a local state S^0 for the environment (not related to any agent in particular): $S = S^0 \times S^1 \times \cdots \times S^n$, hence full observability is usually assumed in this formulation;
- U is the joint action space, composed of local actions for all the agents in the MAS: $U = A^1 \times \cdots \times A^n$. Depending on the problem to be solved, the actions of other agents might be visible or not;
- $T : S \times U \times S \rightarrow [0, 1]$ is the state transition function, which in MAS depends on joint actions instead of local individual actions;
- $R^i : S \times U \times S \rightarrow \mathbb{R}$ is the reward function of agent i, which is now dependent on the state and joint actions; and
- γ is the discount factor.

Since each agent has its own reward function, that is dependent on the other agents, there is not a clear concept defining an optimal policy as in single-agent problems. Simply applying the actions that maximize the local reward may be ineffective if the agents have different reward functions, as one agent might (consciously or not) hamper the performance of another.

Depending on the problem to be solved, the agents can learn as in an MDP ignoring the others (TAN, 1993), try to learn an equilibrium joint policy (HU; GAO; AN, 2015b; HU; GAO; AN, 2015a), or maximize a single common reward (PANAIT; LUKE, 2005). If all agents share a single reward $R^1 = \cdots = R^n$, a central controller that designates actions to each agent can be built by learning through samples of $\langle s, \boldsymbol{u}, s', r \rangle$. However, this solution is unfeasible for most domains because of the requirements in communication and the huge state-action space for the learning problem. The Distributed Q-Learning algorithm (LAUER; RIEDMILLER, 2000) was proposed to solve such problems in a more scalable way. Each agent learns without observing the actions

of the others. However, this algorithm is only applicable in tasks with deterministic transition functions, which is rarely the case for complex tasks.

Equilibrium-based approaches aim at solving the learning problem when agents might have different rewards. For those algorithms, Q-table updates rely on the computation of an equilibrium metric (HU; GAO; AN, 2015a):

$$Q^i_{k+1}(s_k, \boldsymbol{u_k}) \leftarrow (1-\alpha)Q^i_k(s_k, \boldsymbol{u_k}) + \alpha(r^i_k + \gamma\Phi^i(s_{k+1})), \tag{2.4}$$

where Q^i_{k+1} is a Q-table related to agent i, α is the learning rate, and Φ^i is the expected equilibrium value in state s_{k+1} for agent i. Equation (2.4) requires the definition of an equilibrium metric, such as the Nash Equilibrium (HU; WELLMAN, 2003). Therefore, such algorithms are closely related to the Game Theory area, from which efficient equilibrium metrics can be extracted (SODOMKA et al., 2013).

Another popular setting is *adversarial learning*, where the agent has an opponent with diametrically opposed goals. In this case, the optimal policy consists of selecting the action that maximizes the reward supposing that the opponent selected the best action for itself (that is, maximizing the minimum possible return of the actions). For that, the MinMax Q-Learning algorithm (LITTMAN, 1994) can be used, which updates the Q-table as:

$$Q_{k+1}(s_k, a_k, o_k) \leftarrow (1-\alpha)Q_k(s_k, a_k, o_k) + \alpha(r_k + \gamma \max_a \min_o Q_k(s_{k+1}, a, o)), \tag{2.5}$$

where o_k is the action selected by the opponent at step k and o is an action that might be selected by the opponent.

In many MAS applications, each agent might be required to cooperate with a group of agents, while competing against an opposing group. Equilibrium-based solutions are able to generalize to this setting, but it is also possible to treat the opposing team as part of the environment and learn only how to cooperate with teammates.

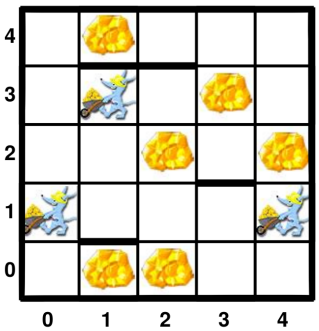
Although those solutions achieved successes, they all require a huge amount of interactions with the environment for achieving a good performance, rendering them hard to scale. Other recent approaches to learn in multiagent RL usually build upon the algorithms discussed in this section to better deal with specific problems, such as non-stationarity (HERNANDEZ-LEAL et al., 2017), still maintaining their scalability problems.

Some models and strategies have been specifically proposed to improve in this

direction. Dec-SIMDPs (MELO; VELOSO, 2011) assume that agent interactions only matter in specific parts of the state space, which means that agents must coordinate in a few states contained in S. Agents act as in a single-agent MDP in the remaining states. CQ-Learning (HAUWERE; VRANCX; NOWÉ, 2010) uses the same idea, finding the states in which coordination is needed through a statistical test. Modeling the task with relational representations is also possible to find commonalities and accelerate learning through abstraction of knowledge (CROONENBORGHS et al., 2005). In the next sections we describe other relevant approaches that scale up RL.

2.3 Object-Oriented MDP

The *Object-Oriented MDP* (OO-MDP) is another MDP extension proposed with the promise to offer generalization opportunities (DIUK, 2009). In order to better explain the OO-MDP concepts, we firstly present the *Goldmine* (DIUK, 2009) domain to provide examples for theoretical definitions. Figure 2.1 illustrates the *Goldmine* domain. There is a certain number of *miners*, which aim to collect all *gold pieces* spread in the environment. There are also impassable *walls* that limit miner movements. At each decision step, all miners may move or collect gold pieces that are close enough. All agents must work collaboratively and the task is completed when all gold pieces in the environment are collected.



(a)

Object *id*	Attributes
*miner*1	$x = 0, y = 1$
*miner*2	$x = 1, y = 3$
*miner*3	$x = 4, y = 1$
*gold*1	$x = 1, y = 0$
	⋮
*gold*6	$x = 4, y = 2$
*wall*1	$x = 1, y = 1,$ $pos = South$
	⋮
*wall*24	$x = 4, y = 4,$ $pos = East$

(b)

Figure 2.1. The *Goldmine* domain. *Miners* aim to gather all *gold pieces* in the environment. Thick *walls* are impassable (adapted from (DIUK, 2009)). (a) Graphical representation. (b) Object representation of the state drawn in (a).

An OO-MDP consists of a tuple $\langle C, O, A, \varrho, D, R\rangle$. $C = \{C_1, \ldots, C_c\}$ is the set of *classes*, where each class C_i is composed of a set of *attributes* denoted as $Att(C_i) = \{C_i.b_1, \ldots, C_i.b_b\}$, and each attribute b_j has a *domain* $Dom(C_i.b_j)$, which specifies the set of values this attribute can assume. The object-oriented representation of the *Goldmine* domain has three classes: *Miner*, *Gold*, and *Wall*, i.e., $C = \{Miner, Gold, Wall\}$. They all have attributes x and y, and walls have an additional pos (position) attribute to indicate the position of the wall in respect to the compass direction: $Att(Miner) = Att(Gold) = \{x, y\}$, $Att(Wall) = \{x, y, pos\}$, $Dom(Miner.x) = Dom(Miner.y) = Dom(Gold.x) = Dom(Gold.y) = Dom(Wall.x) = Dom(Wall.y) = \{0, 1, 2, 3, 4\}$ (in a 5×5 grid), and $Dom(Wall.pos) = \{South, West, East, North\}$. Hence, the set of classes and attributes must be enough to describe all types of objects in the environment and their relevant features. $O = \{o_1, \ldots, o_o\}$ is the set of objects that exist in a particular environment, where each object o_i is an instance of one class $C_j = C(o_i)$, so that o_i is described by the set of attributes from its class $o_i :: Att(C(o_i))$. In an OO-MDP, for each decision step, each object assumes a state given by the current value of all attributes. It may be also important to include an object identification. For example, in the *Goldmine* domain, miners may be identified by a "name", which is used to define the miner that will be moved towards a gold piece. Thus, the object state is defined by the Cartesian product $o_i.state = \left(\prod_{b \in Att(C(o_i))} o_i.b\right) \times o_i.id$, where $o_i.id$ is the object identification.

An object-oriented state is illustrated in Figure 2.1b, where all objects in the environment are described by their attribute values and id. In this case, the state of the underlying MDP is the union of states of all objects $s = \bigcup_{o \in O} o.state$.

The set A consists of actions that may or may not be parameterized. Parameterized actions are defined at the class level $a(params(a)), \forall C_i \in params(a) : C_i \in C$ (i.e., each action affects any object of that class in the same way), thus, parameterized actions are abstract and need to be grounded to be applied. Suppose that an external agent controls all miners and can use the parameterized action *North(Miner)*. This action moves a miner towards North, however, a single miner must be specified in the action parameter in order to apply the action. For example, in the state described in Figure 2.1b, the action *North(miner1)* would move *miner*1 to cell (0,2). ϱ is a set of *terms*, which are boolean functions related to the state transition dynamics in an OO-MDP, which is defined as $\varrho = Rel \cup F$, where $\upsilon : C_i \times \cdots \times C_j \rightarrow Boolean, \upsilon \in Rel$ is a *relation* between objects and F is an optional set of functions defined by the designer to describe domain knowledge. An example of υ in the *Goldmine* domain is the relation $on(Miner, Gold)$, which defines if (and which) miners are in the same position as gold pieces. An example of function to describe domain knowledge that

could be specified is a function *noGold*(), which returns a true value only when all gold pieces in the environment were collected. D is a set of *rules* d, defined as tuples of $\langle condition, effect, prob \rangle$. A *condition* is a conjunction of terms of ϱ and an *effect* e is an operation that changes with probability *prob* the value of attributes in an object, $e : Dom(C_i.b_j) \rightarrow Dom(C_i.b_j)$. As an example of *rule*, consider $d_N = \langle cond_N, e_N, prob_N \rangle$ related to action $North(Miner\ z)$. $cond_N(z)$ verifies if the miner z has a clear path in the north direction and will be able to move in the desired direction, thus $cond_N(z) = \neg touch_N(z, Wall)$, where the relation $touch_N$ returns a true value if the agent sees a wall in the north direction, which makes sure that e_N is only triggered when the miner's movement is not hampered by walls. $e_N(z) = z.y \leftarrow z.y + 1$ changes the position of z towards the desired direction. As *North* is an action with a single deterministic effect, $prob_N = 1$.

Finally, R is a reward function equivalent to the standard MDP one. As we are interested in learning problems, the agent does not know ϱ, D, and R, and has to learn how to actuate through interactions in the environment. The relation between the OO-MDP concepts is depicted in Figure 2.2.

The transition dynamics in an OO-MDP is interpreted as follows. First, at each step k, the agent observes the current state s_k and applies an action a_k. Second, all terms are evaluated to be *true* or $false$ at that step. And third, all rules associated to a_k that had a matched condition trigger an effect that changes attribute values. After all effects have been processed, the change in attribute values results in a state transition, and this procedure can be repeated.

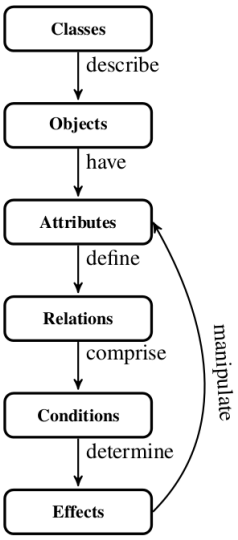


Figure 2.2. Relation between concepts in an OO-MDP (adapted from (DIUK, 2009)).

Even though the OO-MDP allows generalization because of the object-oriented

representation (DIUK; COHEN; LITTMAN, 2008), it takes into account only one agent in the environment.

2.4 Deep RL

As classical RL algorithms achieved successes in practical applications, the community started to push forward RL to applications where listing the complete state-action space is either impossible or infeasible on state-of-the-art hardware. For those problems, explicitly learning a Q function is infeasible, but function approximators might be able to learn a Q function from which a good policy can be extracted.

As *Deep Learning* (LECUN; BENGIO; HINTON, 2015) techniques became more accessible, *Deep Neural Networks* became one of the most popular function approximators for learning RL value functions (MNIH et al., 2015) (which is nowadays called as *Deep Reinforcement Learning* - DRL).

A *Deep Neural Network* architecture is based by multiple layers of neurons (possibly using different activation functions) (SCHMIDHUBER, 2015). Successful applications of DRL include areas where classical RL techniques struggled, such as robotics (LEVINE et al., 2016), physics simulations (DENIL et al., 2017), and Atari games (HESSEL et al., 2018).

Deep Q-Network (DQN) (MNIH et al., 2015), an adaptation of Q-Learning to DRL, is perhaps the first algorithm that can be classified as DRL. The original publication showed that the algorithm can successfully learn policies directly from observing the game-play pixel images while acting in a *Atari game playing* environment. They also demonstrate that the same architecture can learn control policies in a variety of different environments without using prior knowledge and without changing any initial parameter.

The training process of a DQN works similarly as for the original Q-Learning, but now the Q-table update consists in optimizing the neural network by updating the network parameters θ to minimize the following loss function:

$$\mathcal{L}(\theta) = (r + \gamma \max_{a'} Q(s', a', \theta)) - Q(s, a, \theta). \tag{2.6}$$

Very commonly, DRL is carried out by combining the regular RL learning process with additional techniques to stabilize the neural network optimization process, such as maintaining two networks and updating them in a Batch-RL fashion, or using observations multiple times in the optimization process (MNIH et al., 2015).

Although enabling the application of RL in new challenging domains, DRL still suffers from most of the issues of classical RL, especially data-inefficiency. Therefore, scaling up DRL, for which TL can be used, is an open and relevant research area.

2.5 Curriculum Learning for RL

Curriculum Learning typically decomposes a hard (target) task $\mathcal{T}_t$ into several easier (source) tasks $\boldsymbol{\mathcal{T}}_{\mathbb{C}}$. If proper task sequence (*Curriculum*) and TL methods are available, the whole set of tasks may be solved faster than when directly learning $\mathcal{T}_t$, due to the combination of a quick acquisition of knowledge in easier tasks and knowledge reuse.

In Narvekar *et al.*'s description (NARVEKAR et al., 2016), the *Curriculum* is a sequence of tasks within a single domain $\mathcal{D}$. $\mathcal{D}$ has a set of *degrees of freedom* $\mathcal{F}$. Any possible *MDP* that belongs to $\mathcal{D}$ can be built by a generator given a set of values of $\mathcal{F}$, $\tau : \mathcal{D} \times \mathcal{F} \rightarrow \mathcal{T}$. The learning agent is assumed to have a simulator to freely learn in the simpler tasks, and a sequence of source tasks is given by a human[2]. The agent then learns for some time in each of the source tasks specified by the *Curriculum* before trying to solve target task $\mathcal{T}_t$. The steps to build and to use a *Curriculum* are summarized in Algorithm 1. Given the target task $\mathcal{T}_t$, a set of candidates source tasks $\boldsymbol{\mathcal{T}}$ is defined (manually defined in all works so far). Then, the *Curriculum* $\mathbb{C}$ is built or given by a human, specifying a sequence of tasks $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_t), \mathcal{T}_i \in \boldsymbol{\mathcal{T}}_{\mathbb{C}}, \boldsymbol{\mathcal{T}} \subseteq \boldsymbol{\mathcal{T}}_{\mathbb{C}}$, where $\boldsymbol{\mathcal{T}}_{\mathbb{C}}$ is a set of tasks that compose $\mathbb{C}$. All tasks are then solved in order and previous knowledge might be reused for each new task.

Algorithm 1 *Curriculum* generation and use

Require: target task $\mathcal{T}_t$.
1: $\boldsymbol{\mathcal{T}} \leftarrow createTasks(\mathcal{T}_t)$
2: $\mathbb{C} \leftarrow buildCurriculum(\boldsymbol{\mathcal{T}}, \mathcal{T}_t)$
3: $learn(\mathcal{T}_t, \mathbb{C})$ ▷ use $\mathbb{C}$ to learn $\mathcal{T}_t$

Although the strategies discussed in the last sections succeeded in scaling up RL to new applications, reusing previous knowledge is one of the most effective techniques for accelerating learning and can be potentially combined with all those aforementioned techniques. In the next section, we formalize and discuss how knowledge reuse is applicable to RL agents.

[2]Narvekar *et al.* (NARVEKAR et al., 2016) present several heuristic procedures to define a *Curriculum*, but domain-specific human knowledge is required for all of them.

2.6 Transfer Learning

Since learning a task from scratch using RL takes a very long time, reusing existent knowledge may drastically accelerate learning and render complex tasks learnable. As explained in Section 2.2, the learning problem consists of mapping a knowledge space $\mathscr{K}$ to a policy $\pi \in \mathscr{H}$, where $\mathscr{H}$ is the space of possible policies that can be learned by the agent algorithm $\mathscr{A}$, $\mathscr{A} : \mathscr{K} \rightarrow \mathscr{H}$ (LAZARIC, 2012). When learning from scratch, $\mathscr{K}$ consists of samples of interactions with the environment. However, additional knowledge sources might be available. One or more agents[3] may be willing to provide further guidance to the learning agent, such as providing demonstrations of how to solve the problem or explicitly communicating models of the problem or environment. Therefore, the knowledge space is often not only composed of samples from the current (target) task $\mathscr{K}^{target}$, but also of knowledge derived from the solution of previous (source) tasks $\mathscr{K}^{source}$ and from communicating with or observing other agents $\mathscr{K}^{agents}$. Hence, in the general case $\mathscr{K} = \mathscr{K}^{target} \cup \mathscr{K}^{source} \cup \mathscr{K}^{agents}$.

In order to reuse knowledge, deciding *when*, *what*, and *how* to store knowledge into $\mathscr{K}$ and reuse it is necessary (PAN; YANG, 2010). Those three questions are hard and long-studied research problems themselves, and there is no single solution valid for all domains. Unprincipled transfer might cause the agent to reuse completely unrelated knowledge, often hampering the learning process instead of accelerating it (known as *negative transfer*). Therefore, the literature considered many ways to store and reuse (hopefully only) useful information, following varied representations and assumptions.

In general, the main goal of reusing knowledge is to *accelerate* learning. Whether or not a single-agent transfer algorithm learns *faster* than another is commonly evaluated through several of the following performance metrics, summarized by Taylor and Stone (2009) and illustrated in Figure 2.3.

- *Jumpstart*: Measures the improvement in the initial performance of the agent. Since a transfer algorithm might reuse knowledge in $\mathscr{K}^{source}$, successful transfer procedures might initiate learning in a higher performance than when learning from scratch.
- *Asymptotic Performance*: Agents might be unable to reach the optimal performance in complex tasks, converging to a suboptimal policy. TL might help the agents to reach a higher performance, improving their asymptotic performance.

[3] Humans or automated agents (actuating or not in the environment) might be involved in knowledge reuse relations.

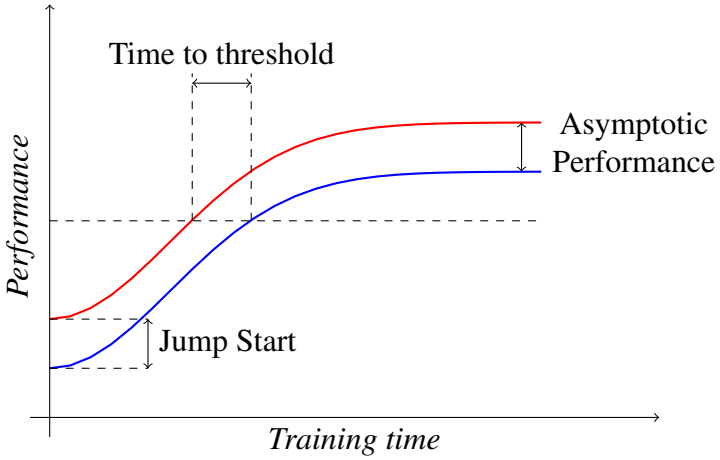


Figure 2.3. Illustration of transfer performance metrics.

- *Total Reward*: In most transfer settings (either when reusing $\mathscr{K}^{source}$, $\mathscr{K}^{agents}$, or both of them), the transfer algorithm is expected to improve the total reward received during training (i.e., the area under the curve of rewards received during the training process).

- *Transfer Ratio*: The ratio between the total rewards received by the two algorithms under comparison (e.g., with and without transfer).

- *Time to Threshold*: For domains in which the agents are expected to achieve a fixed or minimal performance level, the learning time taken for achieving it might be used as a performance metric.

While all those metrics are also generally applicable to MAS, other specific issues must also be taken into account. Communication is a scarce resource in most multiagent applications. Therefore, the overhead in communication demanded by the transfer method cannot be ignored, as well as the computational complexity and more subjective issues such as the restrictions imposed by the assumptions of the transfer method. For example, if a transfer learning method results in a small improvement in performance by the cost of requiring a much higher communication complexity, could this method be considered as effective? The trade-off of all those metrics is usually carefully analyzed in a domain-specific manner.

2.7 Evaluation Domains

In order to evaluate RL algorithms, a sequential decision-making problem must be used to evaluate how much time a learning algorithm takes to learn in it. For compar-

ison purposes, we consider multiple *evaluation domains* throughout this thesis. This section describes the domains we use in the next chapters.

2.7.1 Gridworld

The *Gridworld* domain has been extensively used for evaluations of both single and multiagent TL algorithms. This domain is easy to implement, customize, and to analyze results. Moreover, it is easy to develop simple interfaces that make the task to be solved very intuitive even for laypeople. For this reason, *Gridworld* has become a *first trial* domain for MAS, where methods are validated and tuned before applied in more complex problems.

Basically, any Gridworld domain has a group of agents that can navigate in an environment for solving a task.

While the most classic implementation is a simple navigation task in which the agents aim at reaching a desired destination, popular variations (described in the following subsections) include *Predator-Prey* and *Goldmine*. Figure 2.4 shows the diversity of settings that can be easily built with small variations in the *Gridworld* domain.

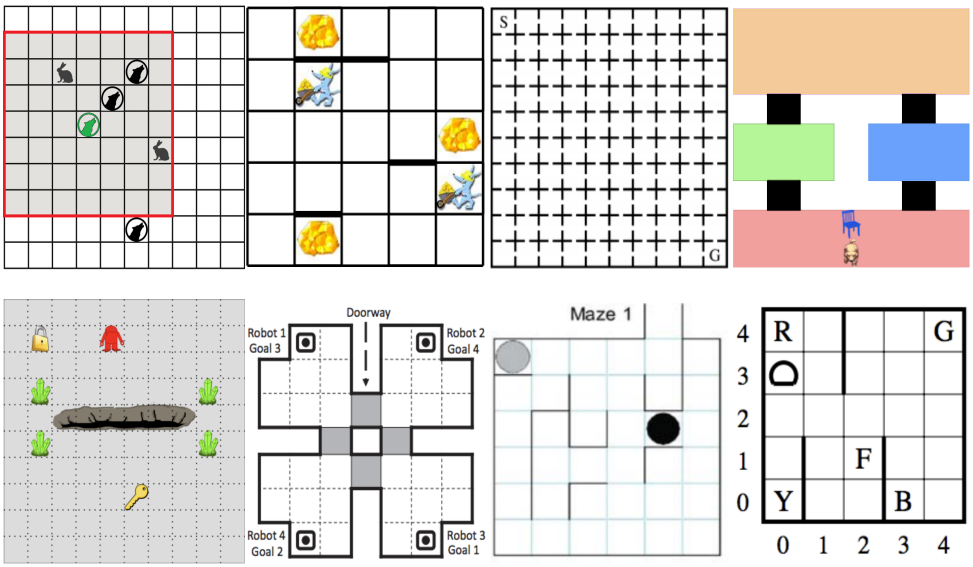


Figure 2.4. Illustrations of different settings of the *Gridworld* domain. From top-left to bottom-right: *Predator-Prey* (SILVA; COSTA, 2017c), *Goldmine* (DIUK, 2009), *Multi-room Gridworld* (KOLTER; ABBEEL; NG, 2008), *Dog Training* (PENG et al., 2016a), *Gridworld with beacons* (NARVEKAR; SINAPOV; STONE, 2017), *Multiagent Gridworld Navigation* (HU; GAO; AN, 2015b), *Theseus and the Minotaur* (MADDEN; HOWLEY, 2004), *Taxi* (DIETTERICH, 2000).

Despite being simple, results observed in this domain allow the evaluation of per-

formance in terms of collaboration and adaptation (e.g., avoiding collisions or collaboratively capturing prey), robustness to noisy sensors and actuators (e.g., including noise in observations or a probability of incorrectly executing chosen actions), and scalability (e.g., how big can be the *Gridworld* that the technique solves?). It is also easy to create more complex versions of a task by trivially increasing the size of the grid or the number of objects in the environment. Moreover, *Gridworld* tasks are very intuitive for humans and can be used for receiving knowledge from laypeople without much effort for explaining the task (PENG et al., 2016a). Therefore this domain will always be an important asset for the development of TL algorithms.

2.7.2 Predator-Prey

In the *Predator-Prey* domain (BOUTSIOUKIS; PARTALAS; VLAHAVAS, 2011; TAN, 1993), predators aim at collaboratively capturing all preys spread in the environment. While each predator is one RL agent in our experiments, preys move randomly at each step (preys could also be considered as reasoning agents, but in our experiments they simply apply random movements regardless of the situation).The environment is depicted in Figure 2.5. At each step, Predators can apply one action to move towards a desired direction $A = \{North, South, West, East\}$. Agents can occupy the same position without punishment, and a prey is captured when it occupies the same position as one predator. In this domain agents cannot observe the whole grid, and their visual field is limited by a *depth* parameter, as shown in Figure 2.5 by the agent with $depth = 3$.

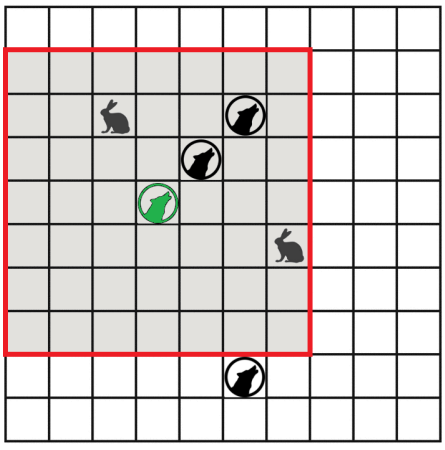


Figure 2.5. Illustration of the Predator-Prey domain. The red square depicts the visual field (depth=3) of the green predator.

Predator-Prey tasks are hard to solve because the environment is non-stationary and partially observable. With an increase in the visual depth the task becomes solvable

in less steps, because preys can be observed from a farther position, however, the state space is increased and it becomes harder to achieve convergence.

2.7.3 Goldmine

Goldmine is a Gridworld variation where a team of *miners* aim at collecting gold pieces in an environment. This domain is more challenging than a navigation Gridworld because the agents have to collaborate and successfully assign which gold they are aiming for at each step to achieve the optimal policy. We mostly use this domain for experiments with object-oriented descriptions. For a complete description of the domain refer to Section 2.3.

2.7.4 Pong

Pong, illustrated in Figure 2.6, is an *Atari* game that has been extensively used as an evaluation domain.

This two-dimensional game consists of controlling an in-game paddle by moving it across the screen to hit a ball towards the opposing side. The learning agent competes against a fixed-strategy opponent. An episode lasts 21 goals, in which a reward of $+1$ is awarded to the player that scores the goal and -1 is given to the other player. Pong is a hard problem to solve, as the game input consists simply of the game screenshot and winning the game requires a sequence of carefully chosen actions.

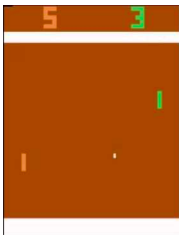


Figure 2.6. Screenshot of the Pong domain.

2.7.5 HFO

The most complex of our evaluation domains is the *Half Field Offense* (HFO) (HAUSKNECHT et al., 2016). This domain is a simplification of the full RoboCup (KITANO et al., 1997) simulated Soccer task. HFO (illustrated in Figure 2.7) consists of a team of friendly agents that must learn how to score goals against a team of highly

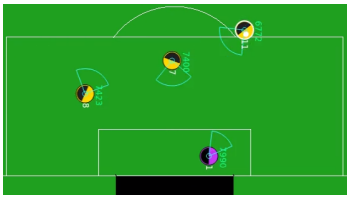


Figure 2.7. The HFO environment with three offensive agents against one goalkeeper.

specialized defensive agents. While HFO is a simplification of the full RoboCup task, it is still a hard learning task in which the agents must take stochastic action effects and noisy observations into account. The agents can benefit from cooperative behaviors, as collaborating with teammates might increase the chance of scoring a goal. A learning episode starts with the agents and ball initiated in a random position in the field, and ends when either the offense agents scored a goal, one of the defending agents caught the ball, the ball left the half field in which the game is played, or a time limit is exceeded.

Defensive players (including the goalkeeper) might be easily configured to follow the Helios policy [4] (AKIYAMA, 2012), which we do in most of the experiments throughout this thesis, while the learning agents start learning from scratch (unless otherwise noted for TL experiments). The agents may communicate but are autonomous, and the following actions are available[5]. When the agent does not have the ball its only option is **Move**, which is an automated action that moves the agent towards the best position guided by the Helios strategy. When the agent is in possession of the ball, it has the following options:

1. **Shoot** – takes the best available shot;
2. **Pass** – passes the ball to a friendly agent;
3. **Dribble** – advances the agent and ball towards the goal.

The agent state is composed of the following observations[6], that are normalized in the range $[-1, 1]$:

1. **agent position**: (X,Y) current agent position on the field;
2. **orientation**: the global direction the agent is facing;

[4] The 2012 RoboCup 2D champion team, a handcrafted expert policy that is unknown by the learning agent.

[5] HFO allows the use of several other variables and action state sets. For reference to all possible options, refer to (HAUSKNECHT et al., 2016)

[6] Notice that in some experiments we make use of only a portion of those features.

3. **ball position**: (X,Y) ball current position;
4. **proximity to the goal**: distance between the agent and the center of the goal;
5. **goal angle**: angle from the agent to the center of the goal;
6. **goal opening**: largest open angle to the goal;
7. **closest opponent proximity**: distance from the agent to the nearest opponent;
8. **positions, goal opening, pass opening, and opponent proximity for each teammate**;
9. **position of each opponent**.

Since the state features are continuous, the domain cannot be directly solved with naive RL algorithms. In some experiments we discretized the state by using Tile Coding (SHERSTOV; STONE, 2005), while we use Deep RL in others.

All the learning agents receive the same reward of $+1$, when the agents score a goal, and a penalty of -1 when the ball leaves the field or when the goalkeeper captures the ball. Otherwise, the reward is 0. The HFO domain has two standard metrics for performance evaluation. *Goal Percentage* (GP) is the percentage of the evaluation episodes in which a goal was scored and *Time to Goal* (TG) is the average number of steps the agent took to score a goal. Ideally, the *Goal Percentage* is as high as possible and the *Time to Goal* is as low as possible (HAUSKNECHT et al., 2016).

Notice that HFO is a complex domain in which agents must take into account continuous variables, partial observability, stochastic effects of actions, and the uncertainty in regard to the opponent's policy.

Chapter 3

Proposed Transfer Learning Taxonomy and Related Work

Although several works have presented evidence that TL might greatly benefit multiagent RL if leveraged properly, the literature is fragmented in methods following different paradigms with little interconnection and flexibility. Integrating different methods is difficult perhaps because no up-to-date work summarized all lines of work in the field and provided a unifying view of the area. Taylor and Stone (2009) provide a comprehensive survey on TL techniques (primarily focused on single-agent RL), but many new approaches have arisen since that survey was published. Lazaric (2012) surveys TL approaches in less depth, proposing a taxonomy to divide the field (also focused on single-agent RL). A handful of surveys focuses on multiagent RL without emphasis on TL. Two major examples are the surveys by Busoniu *et al.* (2008) and Stone and Veloso (2000). While most of the *Learning from Demonstration* and *Inverse Reinforcement Learning* techniques fall within the scope of this survey, Argall *et al.* (2009) and Zhifei and Joo (2012) provide surveys narrowly focused on those topics, which does not provide the aforementioned unifying view.

For this reason, in the course of the work described in this thesis we carefully forged a unified and comprehensive survey of the works in the area. In this chapter, we describe our proposed taxonomy and group a selection of representative recent works. To the best of our knowledge, our work represents the first survey focused on TL for multiagent RL.

Whether or not a transfer procedure should be considered as *multiagent* might prompt debates in some situations. We consider that an agent is composed of its own set of sensors, actuators, and a policy optimizer. Therefore, we included here proce-

dures that: (i) are specialized for reusing the agent's own knowledge across tasks that include multiple agents in the environment; or (ii) transfer knowledge from one agent to another during the learning process. Notice that a human providing demonstrations or guidance during learning is considered a multiagent transfer method, but a designer devising a reward function is not, because the information is defined and made available to the agent before learning.

For sake of brevity, we present here only the common properties of papers under each category and the details of methods that were used as base for our contributions to be described throughout this thesis. A more detailed description of the papers in the area can be found in our main surveys (SILVA; COSTA, 2019; SILVA et al., 2019). Gradual developments of the content of this chapter have been previously published in (SILVA, 2019; SILVA; TAYLOR; COSTA, 2018; SILVA; COSTA, 2017a; SILVA; COSTA, 2016).

3.1 Proposed Taxonomy

TL relies on the reuse of previous knowledge, that can come from various sources. Even though an agent could reuse knowledge from multiple sources simultaneously, in practice the current methods usually focus on a single knowledge source. Hence, we here propose to divide the current literature into two main groups in which we can fit all the TL for MAS publications so far. The methods differ mainly in terms of the knowledge source, availability, and required domain knowledge. We consider the following types of transfer:

- **Intra-Agent Transfer** - Reuse of knowledge generated by the agent in new tasks or domains. An *Intra-Agent Transfer* algorithm has to deal with: (i) Which task among the solved ones is appropriate? (ii) How are the source and target tasks related? and (iii) What to transfer from one task to another? The optimal answer for those three questions is still undefined. Hence, in practice, usually only one of those aspects is investigated at each time, which means that a real-world application would need to consistently combine methods from several publications. We here characterize *Intra-Agent* methods as the ones that do not require explicit communication for accessing internal knowledge of the agents. For example, the procedure of transferring all data from one robot to another similar physical body can be considered as an *Intra-Agent* method. However, if this same information is shared with another agent with an identical body, and this agent tries to merge the transferred knowledge with its own experience, then this

method qualifies as an *Inter-Agent Transfer*. The papers of interest for this survey are the ones that are specialized to multiagent RL (that is, assume that there is at least one opponent/teammate in the environment), or that could be easily adapted for this purpose.

- **Inter-Agent Transfer** - Even though the literature on TL for single-agent RL is more closely related to multiagent *Intra-Agent Transfer* methods, a growing body of methods focuses on investigating how to best reuse knowledge received from communication with another agent, which has different sensors and (possibly) internal representations. The motivation for this type of transfer is clear: if some knowledge is already available in another agent, why waste time relearning from scratch? However, defining *when* and *how* to transfer knowledge is not a trivial task, especially if the agents follow different representations. Some methods focus on how to effectively insert human knowledge in automated agents, while others on how to transfer between automated agents. Nevertheless, comprehensive methods would treat any agent equally regardless of their particularities. Some examples of algorithms within this group are the ones derived from *Imitation Learning*, *Learning from Demonstrations*, and *Inverse Reinforcement Learning*.

In addition to categorizing the papers, we also classify them in several dimensions, according to their *applicability*, *autonomy*, and *purpose*.

The current literature does not offer a method capable of automatically performing all the necessary steps to transfer knowledge in a MAS (both for intra- and inter-transfer). For this reason, most methods focus on a specific subset of problems. Figure 3.1 illustrates how we divide the literature. We use these groupings when discussing the published proposals, and each of them will be explained in details in Sections 3.3 and 3.4.

Notice that the classification given here is not rigid, as many methods share properties with more than one of the categories. Rather than giving a definitive categorization, we here focus on grouping similar approaches to facilitate the analysis of the literature.

In the following subsections, we first discuss the nomenclatures used here and then explain all the considered dimensions for paper classification.

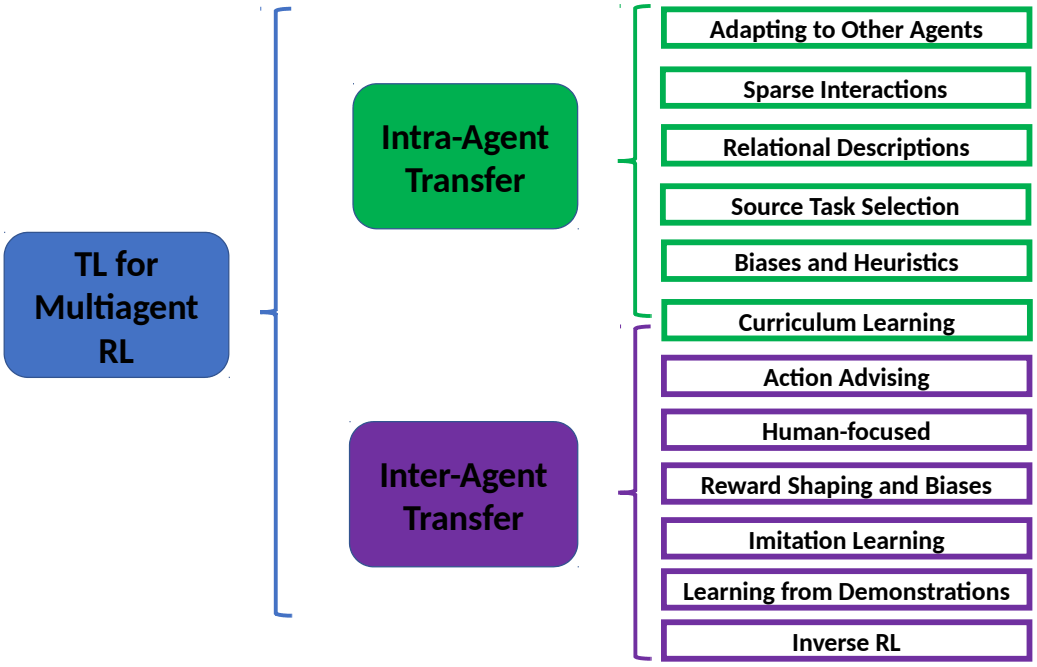


Figure 3.1. Illustration of our classification of the current literature on TL for multiagent RL. Groups to the left are more general and contain the groups to the right.

3.1.1 Nomenclatures

The literature on knowledge reuse has a myriad of terminologies for closely related concepts. Sometimes, multiple terms have the same meaning or the same term is used inconsistently in different publications. In order to avoid confusion, we here discuss our adopted terminology. We do not aim at creating a standard *jargon* to be followed by the whole community, but rather to make clear the distinctions and similarities between the discussed methods.

The literature refers to slightly different settings of the knowledge reuse problem under different names. *Transfer Learning* (TAYLOR; STONE, 2009) is sometimes used to refer to the problem of reusing knowledge across two or more predefined tasks (the target task has as $\mathscr{K}^{source}$ the knowledge gathered in previous tasks). *Multi-task Learning* (FERNANDEZ; VELOSO, 2006) consists of learning multiple tasks at the same time while exploiting similarities between them to reduce the total learning time ($\mathscr{K}^{source}$ is constantly refined with knowledge from the other tasks). *Lifelong Learning* (THRUN; MITCHELL, 1995) aims at consistently reusing knowledge across several tasks that might be presented to the agent during its lifespan ($\mathscr{K}^{source}$ grows over time). *Zero-Shot Learning* (ISELE; ROSTAMI; EATON, 2016) tries to reuse knowledge across tasks without training in the target task (i.e., $\mathscr{K}^{target} = \emptyset$). *Learning from Demonstrations* (ARGALL et al., 2009) focuses on the transfer of knowledge between

agents through explicit communications ($\mathcal{K}^{agents}$ contains demonstrations given by other agents). *Inverse Reinforcement Learning* (ZHIFEI; JOO, 2012) consists in an agent trying to learn a task without access to reward samples, and for that, it tries to estimate a reward function, usually by one agent providing samples of a good policy to another agent. We here use *Transfer Learning* referring to the general challenge of reusing, combining, and adapting knowledge from different sources (agents and tasks).

Regarding the transfer of knowledge between agents, we here adopt the following nomenclatures (first the agent communicating knowledge then the agent receiving it):

- *Advisor/Advisee*: We use this terminology when one agent provides *advice* to another. Here, the knowledge is transferred through explicit communication, but no assumption is made regarding the internal representation or algorithm of other agents. Usually the *advisor* observes the *advisee*'s state and provides information that is expected to be useful in the current situation (e.g., action suggestions). The advising relation might be initiated either by the *advisee* through a help request or by the *advisor* when the *advice* is expected to be most useful. Optionally, advising might be initiated only when both *advisor* and *advisee* agree on initiating the knowledge exchange.

- *Teacher/Student*: A *teacher* agent also transfers knowledge to a *student* through explicit communication. However, here assumptions about the representation or algorithm might be available. Therefore, more information might be possibly communicated, such as value function estimates, models, rules, or demonstrations of some steps following the optimal policy. The received information might refer to the entire state-action space, and not necessarily only to the current situation.

- *Mentor/Observer*: Here, an *observer* agent tries to imitate successful actuation performed by a *mentor* agent. The *mentor* might be aware or not of the *observer*, but no explicit communication happens. Therefore, the *observer* must use its own sensors to implicitly extract knowledge by observing the *mentor*.

3.1.2 Learning Algorithm (LA)

Many MAS algorithms are specialized to a subset of problems (for example, being applicable only to adversarial settings). As some TL methods are associated with the base RL algorithm, they are susceptible to limitations determined by the RL algorithm. We classify LA in one of the following categories:

- **Self-Interested** ($\mathcal{S}$): Those agents apply single-agent RL algorithms taking into account the local state and actions sets. The other agents and their influence are considered as part of the environment dynamics.
- **Equilibrium** ($\mathcal{E}$): Equilibrium-based RL algorithms seek for an equilibrium in the reward functions, usually using game-theoretic approaches to solve the problem. Some TL approaches are based on strategies to reduce the computation of equilibrium metrics, and thus are usable only with these base RL algorithms.
- **Adversarial** ($\mathcal{A}$): Many RL algorithms are specialized to adversarial domains (commonly in a form of zero-sum games). Because of that, some methods have focused on accelerating learning for such adversarial problems.
- **Cooperative** ($\mathcal{C}$): Fully cooperative algorithms assume that all involved agents are benevolent (that is, will never purposely hamper the performance of the system). Therefore, all agents have a common goal. Although not valid for all applications, cooperative algorithms have been broadly used for many problems, especially when the MAS is built by a single owner.

3.1.3 Source Task Selection (ST)

When the agent already has a library of previous solutions and intends to reuse the gathered knowledge in new tasks, a source task selection procedure must be carried out. Since selecting unrelated source tasks may lead to negative transfer, choosing the correct source task(s) is important to successfully reuse knowledge. The source task selection procedure can be trivially carried out by relying on human intuition to manually select it. However, autonomy is desired for many applications, and human intuition may be deceitful as human “sensors" are different from the agent’s. We classify *Intra-agent Transfer* methods in one of the following categories:

- **Implicit** ($\mathcal{X}$) - The approach is only usable inside the same domain (e.g., same action and state spaces with possibly varying goals). Therefore, the source task(s) are commonly reused without concerns in regard to negative transfer.
- **Human-Specified ($\mathcal{H}_s$)** - The agent considers that the source task is manually selected and given as an input to the transfer method, hence a human user/designer is required to perform source task selection.
- **Human-Assisted ($\mathcal{H}_a$)** - A human provides some information to help estimate task similarity (e.g. task parameters). The source task selection is then performed automatically based on this information.

- **Autonomous ($\mathcal{A}$)** - The agent estimates task similarity without additional information, and autonomously select the most promising source task(s).

3.1.4 Mapping Autonomy (MA)

After the source task has been selected, the agent must estimate in which aspects one task is similar to the other. For most applications, applying the entire policy from the source task is either impossible (if the target task has a different state-action space) or will result in a suboptimal actuation. In practice, it is usually more effective to reuse portions of previous solutions but identifying in which aspects the two tasks differ is not a trivial task. We classify the methods in regard to MA in:

- **Implicit** ($\mathcal{X}$) - The approach is only usable for transfer in the same domain, hence the mapping is straightforward.
- **Human-Specified** ($\mathcal{H}_s$) - A human gives an explicit and detailed mapping relating the two tasks.
- **Human-Assisted** ($\mathcal{H}_a$) - A human provides some information to help with the mapping. Some examples of such information are relational descriptions or task parameterizations that might help to relate the tasks.
- **Learned ($\mathcal{L}$)** - The agent creates an automatically generated mapping. This mapping is usually created by estimating a model for each source task and comparing them to a similar one in the target task.

3.1.5 Transferred Knowledge (TK)

Defining *what* to transfer from one task to another is not trivial. As the best transferable information depends on the setting, there is not a single transfer method which is valid for all situations. For example, if one agent transfers information to another and they have different representations, transferring internal structures may be ineffective or impossible, but if the system is composed of homogeneous agents, this same transfer method may be effective. Due to the myriad of settings in which TL has been applied, several possible types of knowledge transfer have been proposed. We classify surveyed proposals in the following categories:

- **Action Advice ($\mathcal{A}_a$)** - If a group of agents has a common understanding of observations and actions, it is possible for them to communicate action suggestions.

Usually, one *advisee* agent communicates its observations to an *advisor*, which can communicate one action to be applied by the *advisee* in the environment.

- **Value Functions ($\mathcal{V}_f$)** - If the agent applies a learning algorithm that uses estimates of value functions to derive policies, it is possible to reuse these estimates across tasks or communicate them to another agent. However, value functions are very specialized for the current task and hard to adapt to similar (yet different) tasks.

- **Reward Shaping ($\mathcal{R}_s$)** - *Reward Shaping* consists of modifying the reward signal received from the environment using additional information to make it more informative to the agent. This information can be originated in a previous task or received from another agent (e.g., another signal received from a human supervisor).

- **Policy (π)** - Task solutions might be transferred across tasks if they are similar enough. Another alternative is to transfer portions of the policy (often called as *options*, *partial policies*, or *macroactions*) to benefit from similarities on parts of the tasks.

- **Abstract Policy (π_a)** - If a relational description is available, an abstracted version of the task solution might be transferred across tasks or between agents. Abstract policies are more general and easier to adapt to possible differences between tasks.

- **Rules ($\mathcal{R}$)** - In addition to being human-readable, rules are easily derived from experience by humans. For that reason, rules have been used to transfer knowledge from humans to automated agents. However, autonomously adapting rules to new tasks is not easy.

- **Experiences ($\mathcal{E}$)** - As explained in Section 2.1, RL agents learn through samples of $\langle s, a, s', r \rangle$ tuples. Those samples can be directly transferred between agents or across tasks. The agent might have to adapt those highly-specialized samples to another task and/or set of sensors though.

- **Models ($\mathcal{M}$)** - During learning, the agent can build models to predict the behavior of other agents or characteristics of the task (such as transition and reward functions). Those models can be reused in new tasks or transferred to other agents if they are able to understand it.

- **Heuristics ($\mathcal{H}$)** - Random exploration (e.g., ϵ-greedy) is very common for RL agents. However, if a heuristic is available, the agent can make use of it to

perform a smarter exploration. Heuristics have been extracted from other agents (most commonly humans) and from previous tasks.

- **Action Set ($\mathcal{A}$)** - For problems in which a large number of actions is available, learning the subset of relevant actions for a problem and transferring it to another might be an interesting way to accelerate learning.

- **Function Approximators ($\mathcal{F}_a$)** - Building an explicit table listing all possible quality values for the entire state-action state is often infeasible (and impossible if continuous state variables exist). In this case, a function approximator is usually iteratively refined after each interaction with the environment. They can also be later reused for new tasks or communicated to other agents, but again they are highly-specialized for the task at hand.

- **Bias ($\mathcal{B}$)** - Instead of reusing the whole policy, the agent might bias the exploration of actions selected by the optimal policy in a previous task. Biases are easier to forget because they quickly lose influence if a bad action is selected.

- **Curriculum (C)** - For many complex tasks, it might be more efficient to divide the complex task into several simpler ones. If appropriate task decomposition and order are available, the agent might be able to learn faster by applying TL from the simpler to the target task.

3.1.6 Allowed Differences (AD)

Assumptions must be made in regard to in which aspects one task may differ from the other. While the simpler TL algorithms assume that the target task is a version of the source task with a bigger state space, ideally the TL method should allow differences in any element of the MDP/SG description and be able to identify in which aspects the two tasks are similar. In practice, how to identify task similarities is still an open problem, hence we classify the proposals in the following categories:

- **Same Domain ($\mathcal{S}$)** - Assumes that the target and source tasks have roughly the same difficulty and are in the same domain (for example, a navigation domain in which only the goal destination can change).

- **Progressive Difficulty ($\mathcal{P}$)** - Assumes that the target task is a harder version of the source task in the same domain, usually with a bigger state-action space due to the inclusion of new objects in the environment, but without significant changes in the transition and reward functions.

- **Similar Reward Function ($\mathcal{R}_f$)** - Assumes that the reward function remains constant or that the optimal policy in the source task still achieves a reasonable performance in the target task. Unlike in *same domain*, here the target task might have a bigger state-action space, as well as different state variables or actions (possibly requiring mappings).

- **Any ($\mathcal{A}$)** - Any aspect of the task might possibly change. The agent must autonomously assess the similarities and discard the source tasks that would result in a negative transfer.

Table 3.1 summarizes all the abbreviations to be used in this Chapter.

Table 3.1. Quick-Reference legend for abbreviations used in Tables 3.3, 3.4, and 3.5.

Learning Algorithm (LA)	*Source Task Selection (ST)*	*Mapping Autonomy (MA)*
$\mathcal{S}$: Self-Interested	$\mathcal{X}$: Implicit	$\mathcal{X}$: Implicit
$\mathcal{E}$: Equilibrium	$\mathcal{H}_s$: Human-Specified	$\mathcal{H}_s$: Human-Specified
$\mathcal{A}$: Adversarial	$\mathcal{H}_a$: Human-Assisted	$\mathcal{H}_a$: Human-Assisted
$\mathcal{C}$: Cooperative	$\mathcal{A}$: Autonomous	$\mathcal{L}$: Learned
Transferred Knowledge (TK)		*Allowed Differences (AD)*
$\mathcal{A}_a$: Action Advice	$\mathcal{V}_f$: Value Functions	$\mathcal{S}$: Same Domain
$\mathcal{R}_s$: Reward Shaping	π: Policies	$\mathcal{P}$: Progressive Difficulty
π_a: Abstract Policies	$\mathcal{R}$: Rules	$\mathcal{R}_f$: Sim. Reward Func.
$\mathcal{E}$: Experiences	$\mathcal{M}$: Models	$\mathcal{A}$: Any
$\mathcal{H}$: Heuristics	$\mathcal{A}$: Action Sets	
$\mathcal{B}$: Biases	$\mathcal{F}_a$ Function Approximators	
$\mathcal{C}$: *Curricula*		

3.2 Methods Proposed in this thesis

Table 3.2 shows a summary of the methods introduced in this thesis and where they fit in the proposed taxonomy.

Table 3.2. Description of our contributions by their categories. Symbols follow the convention of Table 3.1.

Method	Chapter	Gen. Category	Specific Categories
Ad Hoc Advising	4	Inter-Agent ($\mathscr{K}^{agents}$)	**LA**: $\mathcal{S},\mathcal{E},\mathcal{C}$; **ST**: $\mathcal{X}$; **MA**: $\mathcal{X}$; **TK**: $\mathcal{A}_a$
Object-oriented transfer	5	Intra-Agent ($\mathscr{K}^{source}$)	**LA**: all; **ST**: $\mathcal{H}_s$, $\mathcal{H}_a$; **MA**: $\mathcal{H}_a$; **TK**: $\mathcal{V}_f, \mathcal{B}$

3.3 Intra-Agent Transfer Methods

In this section we survey *Intra-Agent* transfer methods. Table 3.3 summarizes the main discussed publications. In the next subsections, we group proposals by their main contributions and, for all the groups discussed, we provide an overview of the current literature.

3.3.1 Adapting to Other Agents

Learning in MAS requires acquiring the ability to coordinate with (or adapt against) other agents. Depending on the task to be solved, agents might have to collaborate with teammates following unknown (and possibly adaptable) strategies, or the environment might allow additions or substitutions of agents at any time (STONE et al., 2010). Nonetheless, the agent still has to cope with the diversity of strategies assumed by other agents and learn how to coordinate with each of them. Therefore, some TL methods focus on reusing experience for learning how to coordinate with new agents faster. In this category, we include methods on how previous knowledge can be leveraged for accelerating coordination with other agents.

As shown in Table 3.3, this group of methods usually assumes that the source task selection is *implicit*. Very rarely (as in the case of (KELLY; HEYWOOD, 2015)), the agent is able to adapt to new tasks, but they are still assumed to be very similar to previous tasks. The main reason for that is the difficulty of the current literature on identifying when and how the strategy of other agents changed, which makes unfeasible to simultaneously account for significant changes in the task. All types of learning algorithms have been explored by this group of methods, and the most common transfer procedure is the reuse of policies for adapting to new agents in the MAS.

3.3.2 Sparse Interactions Algorithms

For some tasks, the actuation of other agents affects the local agent only in a portion of the state space. Thus, it is safe to learn as in a single-agent problem when the actions of other agents do not matter, considerably pruning the joint action space. In this category, we include techniques specially tailored for this setting.

As seen in Table 3.3, this type of TL has been popular for accelerating *equilibrium* algorithms, which are especially benefited from reducing the size of the space in which the equilibrium metric has to be computed. Most of the literature so far has been using these algorithms with the assumption of an *implicit* source task selection, but we be-

Table 3.3. Summary of main recent trends of *Intra-Agent Transfer* methods. We follow the symbols introduced in Section 3.1 and compiled in Table 3.1 for quick reference. The publications are presented in chronological order within their group.

Reference	LA	ST	MA	TK	AD
Adapting to Other Agents (Section 3.3.1)					
(BANERJEE; STONE, 2007)	$\mathcal{A}$	$\mathcal{X}$	$\mathcal{H}_h$	$\mathcal{V}_f$	$\mathcal{R}_f$
(BARRETT; STONE, 2015)	$\mathcal{C},\mathcal{A}$	$\mathcal{X}$	$\mathcal{L}$	π	$\mathcal{S}$
(KELLY; HEYWOOD, 2015)	$\mathcal{S}$	$\mathcal{H}_s$	$\mathcal{H}_s$	π	$\mathcal{R}_f$
(HERNANDEZ-LEAL; KAISERS, 2017)	all	$\mathcal{X}$	$\mathcal{H}_s$	π	$\mathcal{S}$
Sparse Interaction Algorithms (Section 3.3.2)					
(VRANCX; HAUWERE; NOWÉ, 2011)	$\mathcal{E},\mathcal{A},\mathcal{C}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}$	$\mathcal{P}$
(HU; GAO; AN, 2015b)	$\mathcal{E}$	$\mathcal{X}$	$\mathcal{L}$	$\mathcal{V}_f$	$\mathcal{S}$
(ZHOU et al., 2016)	$\mathcal{E}$	$\mathcal{X}$	$\mathcal{H}_s$	$\mathcal{V}_f$	$\mathcal{P}$
Relational Descriptions (Section 3.3.3)					
(PROPER; TADEPALLI, 2009)	$\mathcal{C}$	$\mathcal{X}$	$\mathcal{H}_a$	$\mathcal{F}_a$	$\mathcal{P}$
(KOGA et al., 2013)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{H}_a$	π_a	$\mathcal{S}$
(FREIRE; COSTA, 2015)	$\mathcal{S}$	$\mathcal{H}_s$	$\mathcal{H}_a$	π_a	$\mathcal{S}$
(KOGA; SILVA; COSTA, 2015)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{H}_a$	π_a	$\mathcal{S}$
Source Task Selection (Section 3.3.4)					
(SINAPOV et al., 2015)	all	$\mathcal{H}_a$	$\mathcal{H}_a$	all	$\mathcal{S}$
(ISELE; ROSTAMI; EATON, 2016)	$\mathcal{S}$	$\mathcal{H}_a$	$\mathcal{H}_a$	π	$\mathcal{S}$
(BRAYLAN; MIIKKULAINEN, 2016)	N/A	$\mathcal{A}$	$\mathcal{H}_a$	N/A	$\mathcal{A}$
Curriculum Learning (Section 3.3.5)					
(MADDEN; HOWLEY, 2004)	all	$\mathcal{H}_s$	$\mathcal{H}_a$	$\mathcal{R}$	$\mathcal{P}$
(NARVEKAR et al., 2016)	all	$\mathcal{H}_s$	$\mathcal{H}_s$	$\mathcal{V}_f$	$\mathcal{P}$
(SVETLIK et al., 2017)	all	$\mathcal{H}_s$	$\mathcal{H}_s$	$\mathcal{R}_s$	$\mathcal{P}$
(NARVEKAR; SINAPOV; STONE, 2017)	all	$\mathcal{H}_s$	$\mathcal{H}_s$	$\mathcal{V}_f$	$\mathcal{P}$
(FLORENSA et al., 2017)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{H}_s$	$\mathcal{M}$	$\mathcal{P}$
Biases and Heuristics (Section 3.3.6)					
(BIANCHI; ROS; MANTARAS, 2009)	$\mathcal{S},\mathcal{A},\mathcal{C}$	$\mathcal{H}_a$	$\mathcal{H}_a$	$\mathcal{H}$	$\mathcal{S}$
(BOUTSIOUKIS; PARTALAS; VLAHAVAS, 2011)	$\mathcal{C}$	$\mathcal{H}_s$	$\mathcal{H}_s$	$\mathcal{B}$	$\mathcal{R}_f$
(DIDI; NITSCHKE, 2016)	all	$\mathcal{H}_s$	$\mathcal{H}_s$	π	$\mathcal{S}$

lieve that it would be possible to transfer information about sparse interactions if those algorithms are integrated with source task selectors. The most commonly transferred information is *value functions*, but some of the methods transfer rules defining when the actions of other agents are expected to be irrelevant for defining the optimal policy.

3.3.3 Relational Descriptions

The use of relational descriptions in RL is known to accelerate and simplify learning by generalizing commonalities between states (KERSTING; OTTERLO; RAEDT, 2004; DIUK; COHEN; LITTMAN, 2008). For that reason, relations between objects might also help to find similarities or to perform mappings between tasks. This category includes methods that made use of relational descriptions to transfer knowledge.

Table 3.3 clearly reflects that the main reason for using relational descriptions is for being able to apply *human-assisted* mapping autonomy. The use of relational descriptions also enables the construction of *abstract policies*, which generalize the knowledge obtained and might help to avoid negative transfer. Most of the surveyed methods use *self-interested* learning algorithms, but multiagent relational descriptions could be used for compatibility with other learning algorithms.

3.3.4 Source Task Selection

Before transferring knowledge between tasks, the first step is to select which of the previous solutions will be reused. The selection of source tasks is a very challenging problem, as the agent does not have information on the dynamics of the environment beforehand for computing task similarities. The approaches in this category focus on source task selection.

Table 3.3 shows that source task selection has been largely relying on *human-assisted* mappings so far. The tasks are also expected to have common state-action spaces or to share manually defined task features. Source task selectors are commonly not integrated with knowledge transfer procedures.

3.3.5 Curriculum Learning

Table 3.3 shows that the main characteristic of this group of methods is transferring knowledge across progressively harder tasks. The main focus is usually on how to define task sequences, rather than on how to reuse the knowledge.

Svetlik *et al.* (SVETLIK et al., 2017) propose building the *Curriculum* as a graph, rather than an ordered list (an idea that we will use later in this thesis). The main idea of their proposal is to build a graph of tasks according to a *transfer potential* metric, which estimates how much a source task would benefit the learning process of another. They calculate transfer potential as:

$$v(\mathcal{T}_i, \mathcal{T}_j) = \frac{|Q_{\mathcal{T}_i} \cap Q_{\mathcal{T}_j}|}{1 + |S_{\mathcal{T}_j}| - |S_{\mathcal{T}_i}|}, \tag{3.1}$$

where $v(\mathcal{T}_i, \mathcal{T}_j)$ defines the transfer potential between two tasks $\mathcal{T}_i$ and $\mathcal{T}_j$, $|Q_{\mathcal{T}_i} \cap Q_{\mathcal{T}_j}|$ is the number of Q-values those two tasks have in common[1], and $|S_{\mathcal{T}_i}|$ and $|S_{\mathcal{T}_j}|$ are, respectively, the size of the state spaces in $\mathcal{T}_i$ and $\mathcal{T}_j$. A *Curriculum* graph $\mathbb{C} = \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$, where $\boldsymbol{\mathcal{V}}$ is a set of vertexes (tasks) and $\boldsymbol{\mathcal{E}}$ is the set of edges that define the task sequence, is generated by including all source tasks that have a transfer potential to $\mathcal{T}_t$ higher than a threshold parameter ϵ. Then, $\mathbb{C}$ is used for learning in Algorithm 2 (which fits as a *learn* function in line 3 of Alg 1). First, an unsolved task with indegree *zero* is selected (line 4) and used for training until a stopping criterion is met (line 5). Then, all the edges from the selected task are removed from the graph (line 6) and another untrained task is selected until the target task is solved (that will be the last one). Notice that multiple tasks with indegree zero may exist (line 4); hence more than one task sequence might be generated from the same *Curriculum* depending on the tie-breaking strategy.

Algorithm 2 *learn* in (SVETLIK et al., 2017)

Require: *Curriculum* $\mathbb{C} = \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$, target task $\mathcal{T}_t$.

1: $\boldsymbol{L} \leftarrow \emptyset$ ▷ Learned tasks
2: **while** $\boldsymbol{\mathcal{V}} - \boldsymbol{L} \neq \emptyset$ **do** ▷ While there are unsolved tasks
3: $\quad \boldsymbol{T} \leftarrow \boldsymbol{\mathcal{V}} - \boldsymbol{L}$
4: $\quad$ Select a task $\mathcal{T}_i \in \boldsymbol{T}$ with *indegree* = 0
5: $\quad$ Learn $\mathcal{T}_i$ reusing previous knowledge
6: $\quad$ Remove edges from $\mathcal{T}_i$ in $\boldsymbol{\mathcal{E}}$
7: $\quad \boldsymbol{L} \leftarrow \boldsymbol{L} \cup \mathcal{T}_i$

Notice that every task included in the *Curriculum* must be learned before proceeding to the target task. Therefore, if a big *Curriculum* is given, the agent will probably spend too much time learning source tasks. Moreover, while previous works have shown that building *Curricula* may be beneficial to learning agents, they do not evaluate the advantage of generalization when transferring knowledge between *Curriculum* tasks. To the best of our knowledge the set of source tasks is manually given in all works so far, and an automated source task generation procedure was listed as one of

[1]This computation is domain-specific and must be defined by the designer.

the open problems for *Curriculum* approaches (SVETLIK et al., 2017).

3.3.6 Biases and Heuristics

When learning from scratch, agents must rely on random exploration for gathering knowledge about the environment and task. However, previous knowledge can be used for guiding the exploration of a new task, hopefully increasing the effectiveness of exploration. Biases and Heuristics (Bianchi et al., 2015) are two successful ways for guiding exploration. This category lists multiagent variants of those approaches.

3.4 Inter-Agent Transfer Methods

In this section, we discuss the main lines of research for *Inter-Agent* methods, which represent the majority of TL techniques for MAS. Tan (1993) presented a comprehensive investigation of simple transfer methods applied to cooperative tasks before the *Transfer Learning* term became widely used. His investigation concluded that agents sharing: (i) sensations; (ii) successful episodes; and (iii) learned policies; learn faster than agents individually trying to solve a task. However, all those methods have a very high cost of communication to achieve a speed up. Therefore, more recent transfer procedures try to solve problems with limited communication.

Tables 3.4 and 3.5 depict proposals that represent current research lines. Notice that those proposals assume that all tasks are in the *same domain*. As the focus in those methods is to extract knowledge from communication with other agents, assuming that this knowledge is specific for a single domain is reasonable for most purposes. For this same reason, most of the literature of Inter-Agent Transfer does not perform source task selection and assume that a trivial mapping of the communicated knowledge is enough. We discuss each group in the next sections.

3.4.1 Action Advising

Action Advising is one of the most flexible TL methods. Assuming that the agents have a common understanding of the action set and a protocol for communicating information, one agent might provide action suggestions for another even when the internal implementation of other agents is unknown.

The *teacher-student*[2] framework aims at accelerating training through advice from

[2]Notice that, despite the original nomenclature, this methods fits in our taxonomy as an *advisor-*

Table 3.4. Summary of main recent trends of *Inter-Agent Transfer* methods. We follow the symbols introduced in Section 3.1 and compiled in Table 3.1 for quick reference. Publications are presented in chronological order within their group. For all papers **AD**= $\mathcal{S}$.

Reference	LA	ST	MA	TK
Action Advising (Section 3.4.1)				
(GRIFFITH et al., 2013)	$\mathcal{S},\mathcal{A},C$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(TORREY; TAYLOR, 2013)	C	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(ZHAN; BOU-AMMAR; TAYLOR, 2016)	C	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(AMIR et al., 2016)	C	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(SILVA; GLATT; COSTA, 2017b)	$\mathcal{S},\mathcal{E},C$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(FACHANTIDIS; TAYLOR; VLAHAVAS, 2018)	C	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(OMIDSHAFIEI et al., 2019)	C	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
Human-focused Transfer (Section 3.4.2)				
(MACLIN; SHAVLIK; KAELBLING, 1996)	$\mathcal{S},\mathcal{A},C$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}$
(KNOX; STONE, 2009)	all	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}_s$
(JUDAH et al., 2010)	$\mathcal{S},\mathcal{A},C$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(PENG et al., 2016a)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}_s$
(ABEL et al., 2016)	all	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(MACGLASHAN et al., 2017)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}_s$
(KRENING et al., 2017)	all	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}$
(ROSENFELD; TAYLOR; KRAUS, 2017)	all	$\mathcal{X}$	$\mathcal{H}_s$	$\mathcal{F}_a$
(MANDEL et al., 2017)	all	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
Reward Shaping and Heuristics (Section 3.4.3)				
(WIEWIORA; COTTRELL; ELKAN, 2003)	all	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}_s$
(PERICO; BIANCHI, 2013)	all	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{H}$
(DEVLIN et al., 2014)	C	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}_s$
(BIANCHI et al., 2014)	$\mathcal{A}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{H}$
(SUAY et al., 2016)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}_s$
(GUPTA et al., 2017)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{L}$	$\mathcal{R}_s$
Learning from Demonstrations (Section 3.4.4)				
(SCHAAL, 1997)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
(KOLTER; ABBEEL; NG, 2008)	all	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
(CHERNOVA; VELOSO, 2009)	$\mathcal{S},\mathcal{A},C$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(WALSH; HEWLETT; MORRISON, 2011)	all	$\mathcal{H}_s$	$\mathcal{H}_s$	π
(JUDAH et al., 2014)	$\mathcal{S},\mathcal{A},C$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}_a$
(BRYS et al., 2015)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
(WANG et al., 2016)	$\mathcal{S}$	$\mathcal{H}_s$	$\mathcal{H}_s$	$\mathcal{E}$
(SUBRAMANIAN; JR; THOMAZ, 2016)	$\mathcal{S},\mathcal{A},C$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
(WANG; TAYLOR, 2017)	all	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
(TAMASSIA et al., 2017)	$\mathcal{S},\mathcal{A},C$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
(BANERJEE; TAYLOR, 2018)	C	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$

Table 3.5. Summary of main recent trends of *Inter-Agent Transfer* methods. Second part of Table 3.4.

Reference	LA	ST	MA	TK
Imitation (Section 3.4.5)				
(PRICE; BOUTILIER, 2003)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{M}$
(SHON; VERMA; RAO, 2007)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{M}$
(SAKATO; OZEKI; OKA, 2014)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	π
(LE; YUE; CARR, 2017)	$\mathcal{C}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
Curriculum Learning (Section 3.4.6)				
(PENG et al., 2016b)	all	$\mathcal{X}$	$\mathcal{H}_s$	$\mathcal{C}$
(MATIISEN et al., 2017)	$\mathcal{S}$	$\mathcal{A}$	$\mathcal{X}$	$\mathcal{C}$
(SUKHBAATAR et al., 2018)	$\mathcal{S}$	$\mathcal{A}$	$\mathcal{X}$	$\mathcal{C}$
Inverse Reinforcement Learning (Section 3.4.7)				
(LOPES; MELO; MONTESANO, 2009)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{A}$
(NATARAJAN et al., 2010)	$\mathcal{C}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
(REDDY et al., 2012)	$\mathcal{E}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
(LIN; BELING; COGILL, 2018)	$\mathcal{A}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}$
(CUI; NIEKUM, 2018)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
Transfer in Deep Reinforcement Learning (Section 3.4.8)				
(FOERSTER et al., 2016)	$\mathcal{C}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{M}$
(SUKHBAATAR; SZLAM; FERGUS, 2016)	$\mathcal{C}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{M}$
(DEVIN et al., 2017)	$\mathcal{S}$	$\mathcal{H}_s$	$\mathcal{H}_s$	$\mathcal{F}_a$
(CRUZ; DU; TAYLOR, 2019)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
(OMIDSHAFIEI et al., 2017)	$\mathcal{S},\mathcal{C}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{E}$
Scaling Learning to Complex Problems (Section 3.4.9)				
(TAYLOR et al., 2014a)	$\mathcal{C}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{V}_f$
(KONO et al., 2014)	all	$\mathcal{X}$	$\mathcal{H}_a$	$\mathcal{V}_f$
(XIONG et al., 2018)	$\mathcal{S}$	$\mathcal{X}$	$\mathcal{X}$	$\mathcal{R}$

a more experienced agent (TAYLOR et al., 2014b). The advisor observes the advisee's learning progress and can suggest an action a at any step. However, advice is limited by a *budget* b. After b is spent, the advisor is unable to provide further advice, hence defining *when* to give advice is critical to accelerate learning. In this formulation, the advisor is assumed to have a fixed policy and is able to observe the current state of the advisee. When the advisor identifies a state in which advising is expected to be beneficial, an action is suggested to the advisee, who blindly follows the advice.

The ability to correctly predict relevant states for giving advice is related to the available knowledge regarding the agents in the system. Torrey and Taylor (TORREY; TAYLOR, 2013) propose several heuristics to identify when the advisor should give advice. In the case where the agents use algorithms based on Q-tables, an importance metric can be calculate for a given state. The **Importance Advising** strategy relies on

advisee method

providing advice only for states in which an importance metric $I(s)$ is higher than a predefined threshold. This importance metric is defined as

$$I(s) = \max_a Q^{teacher}(s,a) - \min_a Q^{teacher}(s,a). \tag{3.2}$$

This work also considers the possibility of providing advice only when the advisee's intended action is incorrect according to the advisor's policy, which results in a more efficient use of the budget with the cost of increasing the transmitted data at each step. This strategy is called **Mistake Correcting Advice**. Zhan *et al.* (ZHAN; BOU-AMMAR; TAYLOR, 2016) extended this framework to receive advice from multiple advisors. Their formulation provided the agent with the possibility to refuse to follow a suggested action. However, here too, advisors are assumed to have a fixed policy and only the advisee is learning in the environment. We will later in this thesis relax this assumption and propose a framework in which the agents might be simultaneously learning.

3.4.2 Human-focused Transfer

Although RL algorithms are somewhat based on how autonomous learning is carried out by animals (including humans), actual RL implementations are still very different from the reasoning process of an adult human. Differences on internal representations and how humans and automated agents perceive and interact with the environment impose some obstacles for "transferring" human expertise to an RL agent. Nevertheless, properly receiving guidance and instructions from humans will be required in a world in which machines are integrated into our professional or domestic environments. Moreover, different transfer procedures are usable according to the human technical background. For example, AI experts might be able to design informative reward shapings or rules, while a layperson would probably only be able to give simple instructions, often in a language hard to translate to a machine. Therefore, properly leveraging knowledge from a human is not trivial, and in this category, we include the main current approaches on this line. Notice in Table 3.4 that diverse information might be transferred from a human to an automated agent, and each method tries to make better use of the costly feedback that a human can give to the agent.

3.4.3 Reward Shaping and Heuristics

Exploring the environment is vital for RL agents, as they need to find the best actions in the long-term, which is only possible to compute if all actions have been tried

in all states many times. However, the time taken for randomly exploring the environment is usually prohibitive, and a smart exploration procedure is required. For that, heuristics or reward shapings can be received from *teachers* to improve exploration, hence studying how to build principled heuristics is a very important line of research to make good use of another agent's expertise. Since agents using Potential-Based Reward Shaping (PBRS) are proved to eventually converge to the optimal policy[3] regardless of the shaping quality, those heuristics are some of the few methods that are guaranteed to overcome negative transfer.

3.4.4 Learning from Demonstrations

Learning from Demonstrations is a well-studied category of TL methods in which an experienced *teacher* provides demonstrations to a *student* agent. Demonstrations can be given in many ways, such as teleoperating the *student* or providing some samples of interactions by following the *teacher* policy. The *student* might try to directly learn the *teacher* policy or to use the demonstrations for bootstrapping a RL algorithm. The former achieves good results if the *teacher* policy is close to optimal, but for the general case, the latter is preferred for enabling the agent to learn a better policy than the one used to generate the demonstrations. The main challenge of this category of TL is to leverage the knowledge contained in a limited number of demonstrations. Notice from Table 3.4 that the majority of the discussed methods reuse *experiences*, and that the literature explored all types of learning algorithms.

3.4.5 Imitation

When explicit communication is not available (or other agents are not willing to share knowledge), it is still possible to observe other agents (*mentors*) and imitate their actuation in the environment for acquiring new behaviors quickly. Imitating a *mentor* involves several challenging problems. It might be hard to estimate the performance of another agent online, or maybe even to define which action was applied by the *mentor* (possibly, agents might have different action sets). Finally, imitating another agent might be fruitless if the *mentor* and *observer* have different sensors or learning algorithms in a way that the *observer* is not able to represent that policy. Partly because of those challenges, the literature presents few imitation methods. Table 3.5 shows that those methods extract either *models* or *policies* from observing the mentor.

[3] More specifically, to an optimal policy in an MDP (NG; HARADA; RUSSELL, 1999) or the set of Nash Equilibria in a Stochastic Game (DEVLIN; KUDENKO, 2011).

3.4.6 Curriculum Learning

As discussed in Section 2.5, a *Curriculum* is an ordered list of source tasks intended to accelerate learning in a target task. A promising new research line is the *Transfer of Curriculum*, where one agent builds a *Curriculum* and transfers it to another. Preferably, this *Curriculum* should be tailored to the agent's abilities and learning algorithm. In this category, we list the first investigations on this line, although this challenge has not been solved yet. Notice in Table 3.5 that, as the category name implies, all methods in this group transfer *Curricula* between agents.

3.4.7 Inverse Reinforcement Learning

Although most of the RL techniques assume that rewards can be observed after each action is applied in the environment, *Inverse Reinforcement Learning* (IRL) techniques assume that there are no explicit rewards available, and try to *predict* a reward function through other information made available by agents in the environment, such as demonstrations of the optimal policy. However, estimating a reward through observed instances is an ill-posed problem, as multiple reward functions might result in the same optimal policy. Classical approaches for solving this difficult problem suffer from both a high number of required instances and computational complexity (RAMACHANDRAN; AMIR, 2007), and alleviating those issues is still a topic of current research.

3.4.8 Transfer in Deep Reinforcement Learning

Most of the recent successes achieved by RL in applications rely on function approximators to estimate the quality of actions, where *Deep Neural Networks* are perhaps the most popular ones due to their recent successes in many areas of Machine Learning. Naturally, using DRL introduces additional challenges to train the *Deep Networks* used for function approximation. Especially, their sample complexity requires the use of additional techniques for scaling up learning (e.g., *experience replay*). Therefore, transfer methods especially focused on DRL scenarios might help to scale it to complex MAS applications, since the adaptation of this technique to MAS is still in its first steps (CASTANEDA, 2016; GUPTA; EGOROV; KOCHENDERFER, 2017). Two similar investigations concurrently carried out by different groups evaluated the potential of reusing networks in DRL tasks (GLATT; SILVA; COSTA, 2016; DU et al., 2016). Their results are consistent and show that knowledge reuse can greatly benefit the learning process, but recovering from negative transfer when using DRL might be

even harder.

3.4.9 Scaling Learning to Complex Problems

In this category, we list papers that focused on solving the learning task in high-complexity domains or had promising ideas that can be implemented in the future. Although not presenting novel transfer approaches, the engineering effort to apply techniques already discussed in this paper for challenging domains might help in the development of new TL techniques.

Chapter 4

Simultaneously Learning and Advising

We here describe our contributions on inter-agent teaching methods. We first describe our proposed agent advising framework, called *Ad Hoc Advising*. Then, we describe our first experiments towards extending it to more complex settings. The content of this chapter was previously published in (SILVA; GLATT; COSTA, 2017a; SILVA; GLATT; COSTA, 2017b; SILVA et al., 2019).

4.1 Ad Hoc Advising

Even though the standard Teacher-Student framework (Section 3.4.1) is designed to work only with expert teachers, we here argue that simultaneously learning agents can advise each other and accelerate learning, even when the advisor has not yet converged to a fixed policy, as advice is likely to be useful in the following situations:

1. A learning agent is in a new state (for it), but a friendly agent has already explored that region of the state space.
2. A learning agent is joining a system in which other agents have been learning for some time.
3. The learning algorithm (or internal representation) of an agent might not be as efficient as those of other agents for a given task.

These agents can play both the roles of advisor and advisee during the learning process, as different agents may have explored different areas of the state-action space at a given time step. As all agents may be learning together, the advisor's current policy

is most likely not optimal. In fact, even the assumption that the advisor performs better than the advisee may be unrealistic if the agents blindly advise each other. Hence, agents must be able to evaluate how confident they are in their current policy. We here propose a new framework for advice taking in which multiple simultaneously learning agents can share advice between them. To the best of our knowledge our proposal is the first policy advice framework intended to accelerate learning in a MAS composed of simultaneously learning agents. Our work differs from (NUNES; OLIVEIRA, 2003) because in our setting all agent rewards depend on the *joint* actuation of all agents. Furthermore, as all agents may be learning at the same time unlike in the majority of the advice taking proposals, we do not assume that any agent follows a fixed policy. In our proposal, advice takes place in jointly-initiated *advisor-advisee* relations (i.e., these relations are only initiated under the agreement of both the advisor and advisee (AMIR et al., 2016)), which are established on demand when the advisee is not confident in its policy in a given state and one or more advisors believe that they have more knowledge for that state. All agents are constrained by two budgets. The first one limits the number of times an agent can receive guidance, and the second one limits the number of times it can advise other agents. Our proposal is described in the next sections.

4.1.1 Advice Strategy for Simultaneously Learning Agents

We are interested in MAS composed of multiple autonomous agents in a shared environment, where an agent can learn by exploring the environment and also accelerate the learning by asking for advice from another agent that already has a better policy for the current state. Here we focus on MAS composed of multiple RL agents, although our framework is formulated more general and applicable for agents using any learning algorithm.

Unfortunately, identifying which of the agents have a good policy is not easy, since some (or all) of them may be simultaneously learning in the same environment. Hence, instead of providing a fixed teacher (or set of teachers) like in the previous works, we propose to build *ad hoc advisor-advisee* relations. These relations are established for a single step according to each agent's confidence in its own policy for the current state. Each agent in our advising framework is equipped with a tuple $\left\langle P_{ask}, P_{give}, b_{ask}, b_{give}, G, \Omega \right\rangle$ in order to be part of *advisor-advisee* relations. $P_{ask} : S \rightarrow [0, 1]$ is a function that maps a given state to the probability of asking for advice. At each step, before choosing their action for the current state, agents consult their P_{ask} function, and broadcast a request for advice to all reachable agents

with the calculated probability. G defines the set of reachable agents for a given state[1] $P_{give} : S \rightarrow [0, 1]$ encodes the confidence of the agent in its own policy and represents the probability of giving an advice when requested. P_{ask} and P_{give} change over time. P_{ask} is expected to output smaller probabilities with increasing training time, while P_{give} is expected to output higher probabilities over time. b_{ask} and b_{advice} are, respectively, the budgets to receive and to give advice, where the former is the maximum number of times an agent can receive advice and the latter is the maximum number of times an agent can give advice. Finally, Ω is a function that combines the received advice (if more than one agent answered the request) and selects one of the actions.

We rely on a *jointly-initiated* teaching strategy, as proposed by Amir *et al.* (2016), to define when an *advisor-advisee* relation is initiated, which happens only when an advisee i decides to ask for advice in a given time step (according to a probability defined by P_{ask}) and an advisor j decides to answer the advice request (according to P_{give}). An *advisor-advisee* relation is composed of $\left\langle i, j, s^i, \zeta, \pi^j \right\rangle$, where s^i is the state from the advisee's point of view, ζ is a function to translate s^i to a state that j is able to understand, and π^j is the advisor's policy (to extract the advice). In a fully observable environment, ζ transforms the state s^i to exchange the state features of the advisee with the advisor without the need for additional communication. In a partially observable environment, the advisee needs to communicate its observations, which are processed by ζ as if they were the advisor's observations. The advisor then advises the action according to $\pi^j(\zeta(s^i))$ and terminates the relation. Since advisor policies are directly used, we assume that either the agents are members of a team that receive the same reward ($R^1 = \cdots = R^m$) or that the rewards of both agents are the same for a given tuple of local observations and the joint actuation.

At every learning step, the agent sends a broadcast to all reachable agents asking for advice in case it is not confident enough in its policy for the current state. Then, the agents who receive the call for advice evaluate their confidence for the received advisee state and establish an *advisor-advisee* relation if they are confident enough in their policy for that state. The advisee combines all received advice, executing one of the suggested actions. If no advice is provided, the agent follows its usual exploration strategy. All agents are limited by a budget to ask for advice b_{ask} and a budget to give advice b_{give}. As the actions are directly extracted from the advisor policy, we assume, that, in all *advisor-advisee* relations $\pi^j(\zeta(s^i)) \in A^i$, i.e., in a MAS composed of heterogeneous agents, all communicated actions must be available to the advisee.

We here assume that the agents are cooperatively trying to solve a task. Even

[1] Some agents in the system may be temporarily unreachable, as communication may only be possible if the agents are near, for example.

though our framework can be used to work with self-interested agents, that would require the development of trust and/or argumentation mechanisms to identify prospective advisors in each state, which is outside of the scope of this work.

Algorithm 3 fully describes the action selection in our framework. The agent first observes the current state s^i. As long as it still has a budget b_{ask} to ask for advice, it calculates the probability p_{s_i} of doing it in the current state according to P_{ask} and then, if so chosen, sends a broadcast message to all reachable agents. If the agent receives advice from any adviser, its budget b_{ask} is decremented and it executes the provided action. The Ω function determines which advice to follow in case more than one agent responded to the call. Finally, if no advice is received (either because the agent did not ask for it or because no agent answered), the usual exploration strategy is executed.

Algorithm 4 describes how an agent decides if it is confident enough to give advice for the current state if it still has an available budget to do so. At every received call for advice, the agent estimates the probability p_{s_i} for giving advice following P_{give}, representing the confidence for its policy in the current state. If it chooses to give advice, it decrements its advice giving budget b_{give} and establishes an *advisor-advisee* relation to suggest an action according to its policy. If it chooses not to give advice, the call is simply ignored. All agents in the MAS may play both the roles during the learning process. We assume that the agents are solving a cooperative task and can understand or observe the state of one another, while being able to use different internal representations or algorithms. As in the before mentioned proposals, the ability of predicting the states in which advice is useful is critical to the success of the method. However, in our setting, an importance metric $I(s)$ calculated as in Equation (3.2) is likely to be misleading because the Q-values can vary significantly until the agents have converged their policies due to the random exploration order.

4.1.2 Deciding When to Ask for and When to Give Advice

Deriving the **Importance Advising** for our proposal is not straightforward, because the importance metric $I(s)$ of Equation (3.2) may be misleading for our setting as it does not evaluate the agent confidence in the current Q-value estimate. Thus, we propose a new importance metric that takes into account how many times the agent explored the desired state. With the assumption that the Q-value estimate becomes more reliable when the environment is repeatedly explored, we calculate the advice probability to ask for or give advice as:

$$P_{ask}(s, \Upsilon) = (1 + \upsilon_a)^{-\Upsilon(s)} \tag{4.1}$$

Algorithm 3 Action selection for a potential advisee i

Require: advice probability function P_{ask}, budget b_{ask}, action picker function Γ, confidence function Υ.

for all training steps **do**
  Observe current state s^i.
  if $b_{ask} > 0$ **then**
    $p_{s_i} = P_{ask}(s_i, \Upsilon)$
    With probability p_{s_i} **do**
      Define reachable agents $G(s^i)$.
      $\Pi \leftarrow \emptyset$
      for $\forall z \in G(s^i)$ **do**
        $\Pi = \Pi \cup z.ask\_advice(s^i)$
      if $\Pi \neq \emptyset$ **then**
        $b_{ask} \leftarrow b_{ask} - 1$
        $a \leftarrow \Omega(\Pi)$
        Execute a.
  if no action was executed in this step **then**
    Perform usual exploration strategy.

$$P_{give}(s, \Psi) = 1 - (1 + \upsilon_g)^{-\Psi(s)}, \tag{4.2}$$

where Υ and Ψ are functions that evaluate the confidence for the current state to, respectively, ask for and give advice ($\forall s \in S, \Upsilon(s) \geq 0$ and $\Psi(s) \geq 0$). υ_a and υ_g are scaling variables. These equations were designed to return a higher probability of asking for advice when the agent has a low confidence in the current state. Similarly, the probability of providing an advice is higher when the confidence in the state is high. Thus, a higher υ_a value results in a lower probability of asking for advice, while a higher υ_g results in a higher probability of giving advice. We here propose two confidence functions, that may be used depending on the characteristics of the agents in the MAS. The first metric can be used for any kind of agent, regardless of which learning algorithm is used. With the assumption that the agent is learning in the environment and its policy is improving with the learning process, we calculate the confidence as:

$$\Upsilon_{visit}(s) = \sqrt{n_{visits}}, \tag{4.3}$$

where $n_{visit}(s)$ is the number of times the agent visited the state s. As the agent repeatedly visits a given state, this function returns a higher value and, consequently, the probability for asking for advice is lower. Υ_{visit} is also appropriated for deciding when to give advice when the learning agent algorithm is variable or not known in advance, $\Psi_{visit}(s) = \Upsilon_{visit}(s)$. If the agent can be assumed to follow a *temporal difference*

Algorithm 4 Response to advice requirement (called as *ask_advice*).

Require: advice probability function P_{give}, budget b_{give}, advisor policy π, advisee state s^i, state translation function ζ, confidence function Ψ.

1: **if** $b_{give} > 0$ **then**
2: $\quad p_{s_i} = P_{give}(s^i, \Psi)$
3: $\quad$ **With** probability p_{s^i} **do**
4: $\quad\quad b_{give} \leftarrow b_{give} - 1$
5: $\quad\quad$ **return** $\pi(\zeta(s^i))$
6: **return** $\emptyset$

learning algorithm, we can use the second confidence function, computed as:

$$\Psi_{TD}(s) = \Upsilon_{visit}(s)\ |max_a Q(s,a) - min_a Q(s,a)|. \tag{4.4}$$

This function takes into account both the number of times the agent visited the current state (encoded by Υ_{visit}) and the importance of giving an advice in that state (as in Equation (3.2)).

Thus, we derive the **Visit-Based Advising** (Υ_{visit}, Ψ_{visit}) and **Temporal Difference Advising** (Υ_{visit}, Ψ_{TD}) for our setting by following these confidence functions.

In case the advisee receives more than one advice, we select the executed action through a majority vote, as in (ZHAN; BOU-AMMAR; TAYLOR, 2016).

In the next Section we present an experimental evaluation of our proposal.

4.1.3 Experimental Evaluation

The teacher-student framework (TORREY; TAYLOR, 2013) was originally devised for single-agent tasks, but can be modified to be usable in our setting. Tan's (TAN, 1993) proposal can also be easily adapted. Thus, we investigate the following advising strategies in our experiments:

- **Ad Hoc temporal difference advising (AdHocTD)** - Our proposal is implemented as described by (Υ_{visit}, Ψ_{TD}) in the previous section when the agents can be assumed to follow a temporal difference learning approach (and thus the estimated Q-value is available);

- **Ad Hoc Visit-Based advising (AdHocVisit)** - We also compare the results of our proposal when the fitness of states for advising is evaluated only through the number of visits, as described by (Υ_{visit}, Ψ_{visit}) in the previous section (i.e., learners cannot be assumed to follow a temporal difference approach);

- **Importance-Based Teacher-Student Advising (Torrey)** - We implement the original teacher-student framework for our setting by defining each agent in the system as a student for another agent as teacher. As in the original formulation, advice is given when the teacher detects a state for which Equation (3.2) is greater than a predetermined threshold t. As in our proposal, the agents rely on a state translation function ζ to make correspondences between the teacher and student states;

- **Episode Sharing (EpisodeSharing)** - As proposed in (TAN, 1993), the agents share tuples of $\langle s, a, s', r \rangle$ after a successful episode. However, here the agents are restricted by a budget, from which one unit is reduced for each shared tuple;

- **Regular Learning (NoAdvice)** - As reference, we also show the performance of the $SARSA(\lambda)$ learning algorithm without advice.

Note that this adaptation of the standard teacher-student framework for our setting assumes that the teacher is able to evaluate the student state at all time steps. Futhermore, the teacher-student relation is fixed for a pair of agents and our proposal is robust to states or situations in which the advisor is not able to evaluate the advisee state. Multiple agents may provide advice for the same state and all advising relations are dynamic. We perform an experiment to evaluate how the different advising techniques affect the learning of a complex multiagent task. The HFO (Section 2.7.5) environment is our experiment domain. We performed all the experiments using $SARSA(\lambda)$ with a learning rate of $\alpha = 0.1$, a discount factor of $\gamma = 0.9$, and decay rate of $\lambda = 0.9$. The ϵ-greedy strategy was used as the exploration strategy of all agents with $\epsilon = 0.1$.

In order to evaluate the learning speed of the agents with each of the advising frameworks, we trained the agents for 5000 episodes, and evaluated the agents' current performance every 20 episodes. During evaluation, the Q-table update, random exploration, and the ability to ask for advice are turned off for all agents, and the current policy is used to play 100 episodes in order to evaluate the performance of all algorithms. We performed experiments considering two scenarios. On the first one, all three agents are trained from scratch. On the second scenario, one of the agents was trained for 3000 learning steps before the learning process. We also evaluated our approach using the **Mistake Correcting Advice** strategy (TORREY; TAYLOR, 2013) in the first scenario. The *Mistake Correcting Advice* assumes that the agent asking for advice can communicate more data, and also informs its intended action together with the state. We expect that in this situation our proposal will be able to present the same performance while expending less budget.

While the first scenario is of more interest for us, because all agents are simultaneously learning, the second takes into account systems where some of the agents are already trained, and we expect that the previous state-of-the-art methods will perform better in the second one.

The results of the next Section are averages over 50 executions of this procedure. In all experiments, we used $b = 1000$ and $t = 0.01$ for *Torrey* and *EpisodeSharing*, and $b_{ask} = b_{give} = 1000$ for *AdHocVisit* and *AdHocTD*. $\upsilon_a = 0.5$ for both *AdHocVisit* and *AdHocTD*, while $\upsilon_g = 0.5$ for the former and $\upsilon_g = 1.5$ for the latter. A budget of 1000 steps corresponds to approximately 10 successful episodes receiving guidance for all steps. The statistical significance of the experiments were assessed through a 95% confidence Wilcoxon signed-rank test.

We present the results for all evaluated scenarios in the next Sections.

All Agents Learning from Scratch

Figure 4.1 (top-left) shows the *Time to Goal* metric for all algorithms. During the first learning episodes the agents score goals very quickly because they are attempting long shots right after obtaining the ball possession. When this behavior results in a goal, TG is very low, however, this happens rarely, which can be seen by the low GP observed in Figure 4.1 (bottom) on the initial episodes. After roughly 700 episodes, the agents were able to find a safer behavior to score goals, resulting in higher TG values yet allowing an increase in GP. After 3000 learning episodes, the difference between the TG observed for *AdHocVisit* and *AdHocTD* became statistically significant when compared to *NoAdvice*. *Torrey* had a significantly lower TG than *NoAdvice* until roughly 800 learning steps and after that maintained comparable results. Finally, *EpisodeSharing* was not significantly better than *NoAdvice* in this metric. Thus, the *ad hoc* advising achieved the best results regarding TG. However, as the difference in average is not very high (roughly 3 steps), we focus on the GP metric to compare the algorithms both in this scenario and in the following ones.

Figure 4.1 (bottom) shows the GP observed for all algorithms and Figure 4.1 (top-right) shows the spent budget. As expected, *Torrey*'s importance metric was misleading because the agents were learning from scratch, which caused a significantly worse performance (when compared to *NoAdvice*) between (roughly) 100 and 750 learning episodes, interval in which all the budget was inefficiently spent. After that, the difference was not significant, which means that Torrey brought no benefits at all in this experiment.

In his work, Tan (TAN, 1993) concludes that *EpisodeSharing* accelerates learning when the agents have an unrestricted budget. However, in this experiment all the budget was spent after 80 learning episodes. This resulted in a speed-up between 600 and 800 episodes, but after that the results were not significantly different from *NoAdvice*, which shows that *EpisodeSharing* does not perform well when a restricted budget must be taken into account. *AdHocVisit* was significantly better than *NoAdvice* between 1400 and 1900, and in all evaluations after 4000 learning episodes, resulting also in a better asymptotic performance. Notice also that *AdHocVisit* finished the learning process without using all the available budget, which means that the advice was thoughtfully spent. Finally, *AdHocTD* was significantly better than *NoAdvice* from 1400 learning episodes until the end of training. While *AdHocTD* was never significantly worst than any of the algorithms after 1000 training steps, it also finished the experiment using less than half the available budget. The results of this experiments show that *AdHocTD* is an efficient advising method when all agents are learning simultaneously, and that *AdHocVisit* can achieve a reasonable speed-up when the learning algorithm is unknown.

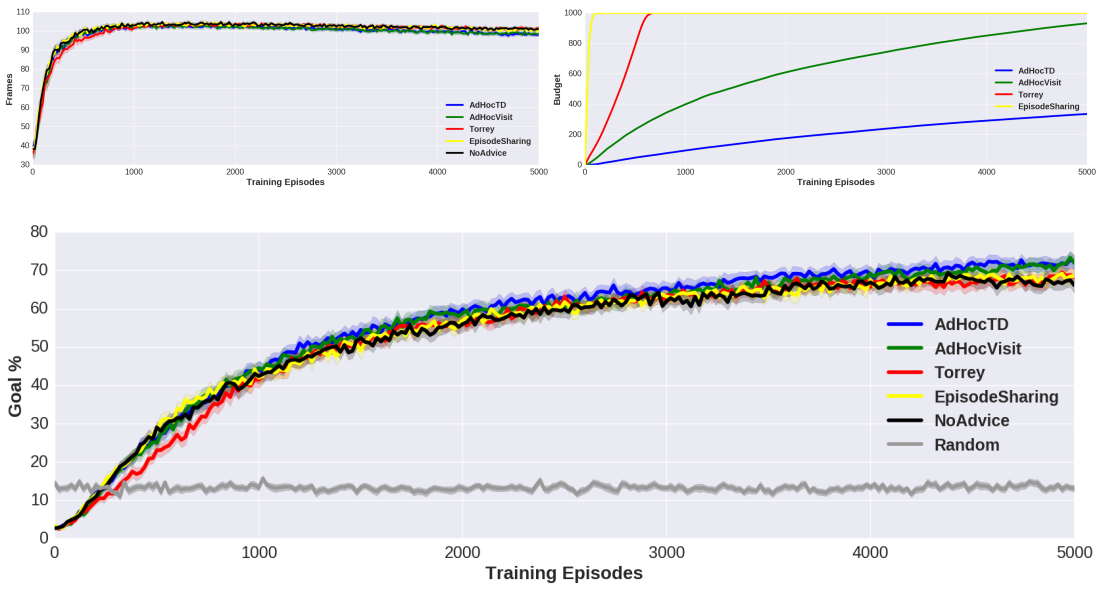


Figure 4.1. The average number of steps to score a goal (top-left), average spent budget (top-right), and percentage of goals (bottom) observed in the evaluation steps for each algorithm learning from scratch. The numbers are averages from 50 runs and the shaded area corresponds to the 95% confidence interval. The performance achieved by a random agent was included as baseline.

Correcting Advice

We here give the ability to the advisee to communicate its intended action together with the current state when asking for advice. The previous experiment is repeated, and we show the results for AdHocTD, AdHocVisit, and Torrey (that are the algorithms for which the Mistake Correcting Advice can be implemented). Figure 4.2 shows the GP observed for these algorithms. *AdHocVisit* and *AdHocTD* present no significant difference in performance when comparing to the results in Figure 4.1. However, when comparing the spent budget (Figure 4.3a) it is noteworthy that the same performance is achieved using less budget, which means that the *ad hoc* advising makes good use of the extra available information. On its turn, Torrey presents negative effects in this scenario. Figure 4.3a shows that the extra information allowed the algorithm to use the budget for more time. However, this only increased the number of misleading advice, resulting in a significantly worse performance after 2700 episodes, and in a worse asymptotic performance.

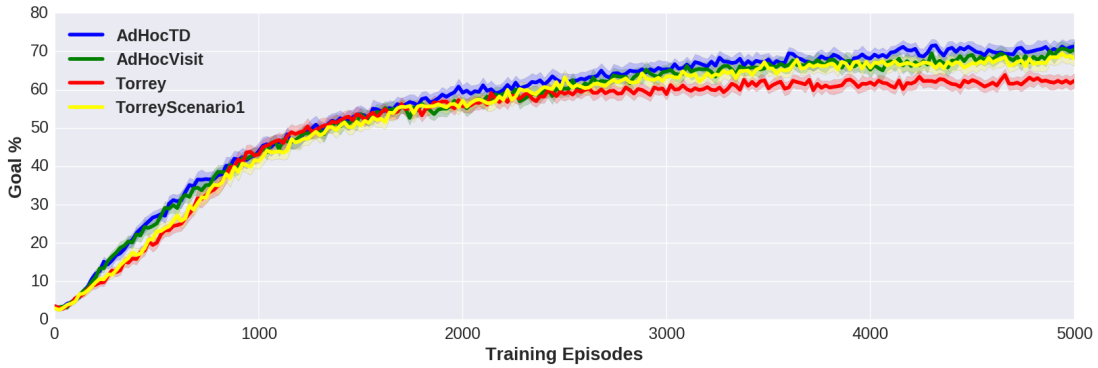
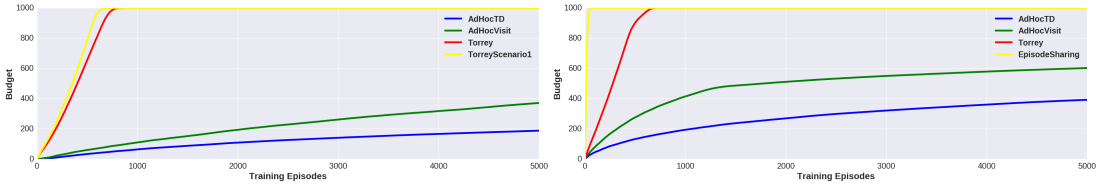


Figure 4.2. The average percentage of goals observed in the evaluation steps for the *Mistake Correcting Advising*. The numbers are averages from 50 runs and the shaded area corresponds to the 95% confidence interval. The Torrey performance without sharing the intended action was included as a baseline.



(a) **(b)**

Figure 4.3. The average spent budget per evaluation steps for each algorithm in the experiments of this section.

One Expert Agent Included

Figure 4.4 shows the GP metric observed for all algorithms, while Figure 4.3b shows the spent budget. Notice that all the algorithms (including NoAdvice) performed better than when all agents are learning from scratch, which means that the agents learn to collaborate and solve the task faster when an expert is present. Torrey is significantly worst than NoAdvice from 120 learning episodes until roughly 650 episodes. Even though the expert now offers good advice, sometimes the two learning agents present misleading advice, which still hampers learning. After that, Torrey is not significantly better than NoAdvice except for a short interval around episode 4000. On the other hand, EpisodeSharing was benefited from the presence of an expert agent, and presented a significantly better performance than NoAdvice from roughly 350 to 550, 2600 to 3000, and 4950 to 5000, also finishing the learning process with a better asymptotic performance. AdHocVisit was worse than NoAdvice from 100 to 400 learning episode, but after 1750 episodes AdHocVisit was always better and achieved a higher asymptotic performance. The spent budget is also smaller than in the first scenario, because the expert agent seldom ask for advice. AdHocTD was never significantly worse than NoAdvice, and was always better after roughly 1300 learning trials. Also, AdHocTD spent less budget than AdHocVisit. Compared to other Advising frameworks, AdHocVisit was worse than EpisodeSharing until episode 800. After that, AdHocVisit achieved a better result for almost all evaluations after episode 2800. AdHocTD was never significantly worse than EpisodeSharing and always better after episode 2700. Finally, AdHocTD was never worst than AdHocVisit and had a better performance for most evaluations since the beginning of the learning process.

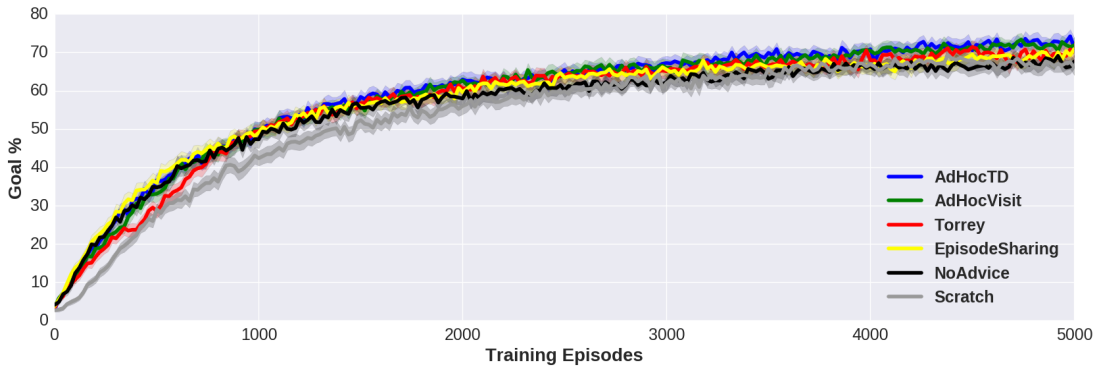


Figure 4.4. The average percentage of goals observed in the evaluation steps for each algorithm when one agent is an expert. The numbers are averages from 50 runs and the shaded area corresponds to the 95% confidence interval. The NoAdvice performance when all agents are learning from scratch was included as a baseline.

4.1.4 Discussion

These scenarios showed that the *ad hoc* advising presents a better performance than other state-of-art algorithms when multiple agents are simultaneous learning in a shared environment. For most cases when our advising strategy was significantly better the percentage of score goals was approximately 6% higher in average, which is a significant speed-up when taking into account the complexity of the domain, the strict limitation in the number of communications for asking for advice, and that the agents have no access to previous information as supposed by many Transfer Learning algorithms.

In addition to presenting a better performance, our proposal makes use of less advising interactions, spending less budget than previous proposals in our setting. We showed that the used budget can be further reduced by using the *Mistake Correcting Advice* strategy, i.e., including the intended action in communications when asking for advice.

Finally, our experiments indicate that, when the learning agents can be assumed to follow temporal difference algorithms, the *AdHocTD* formulation achieves a better performance, but the *AdHocVisit* formulation still presents reasonable improvements in the learning process without assumptions in regard to the learning algorithm.

4.2 Scaling up Ad Hoc Advising

The main limitation for applying *Ad Hoc Advising* together with *Deep Reinforcement Learning* (MNIH et al., 2015) would be that both confidence functions proposed in Section 4.1.2 require a state counter, which cannot be recovered from the function approximator. As discussed in the previous section, even though previous frameworks do not use this counter, they are not applicable because all the agents are simultaneously learning. Therefore, the first (and probably most important) improvement in our proposal would be a confidence function that could be extracted from general-purpose RL algorithms that use function approximation. For that purpose in this subsection we explore a way to estimate the agent confidence directly through the function approximator. This approach can be combined with Ad Hoc Advising or used in other scenarios where action advising is desirable. We propose *Requesting Confidence-Moderated Policy advice* (RCMP), an algorithm to selectively give advice to a learning agent in situations where its epistemic uncertainty[2] is high. In contrast, when the agent has a

[2]The uncertainty stemmed from lack of information about the environment the agent is trying to model. On the other hand, *aleatoric* uncertainty comes from the environment stochasticity. The former

low uncertainty (assuming that the uncertainty estimation is accurate), this indicates that the value estimate for the current state is close to convergence, and advice might be saved for more useful situations. RCMP is compatible with Deep RL techniques, could be combined with Ad Hoc Advising, and can be easily implemented with any contemporary ML framework. The content of this section was previously published in a workshop paper (SILVA et al., 2019).

4.2.1 Uncertainty-Aware Advice for Deep RL

We are interested in leveraging action advice to accelerate the learning process, while allowing the agent to explore the environment and improve upon the demonstrator's policy. For that purpose, we propose the *Requesting Confidence-Moderated Policy advice* (RCMP) algorithm.

We assume that a demonstrator $\pi_\Delta : S \times A \rightarrow [0, 1]$ is available to the agent and can be queried to give action suggestions (we use $\pi_\Delta(s)$ for getting an action sample from π_Δ for state s). While π_Δ might follow any algorithm, the learning agent has no knowledge about the internal representation of π_Δ and can only get samples of $\pi_\Delta(s)$. Furthermore, we assume that π_Δ has a policy that performs significantly better than a random policy. The demonstrator might be unavailable at some times, e.g. if the human will be participating in the learning process only for a short period of time, hence the learning agent is equipped with an availability function $\boldsymbol{A}_t$ to check if the demonstrator is available at a given step t. Once the budget is spent, the demonstrator is unavailable for the rest of training. The availability can also be evaluated in a domain-specific way, e.g. considering the demonstrator as unavailable when the physically distance between the agents is too high, and this obstructs their communication. In most cases, the demonstrator can neither be assumed to be available at all times nor to have an optimal policy.

Algorithm 5 fully describes RCMP. The learning agent might use any value-function-based algorithm as long as it is able to estimate an epistemic uncertainty measure μ from its model of the value function (we propose a way of adapting DQN-like algorithms in the next section). Firstly, the agent initializes the Q-function $\hat{Q}$ and the policy π (line 5). Then, for every learning step, the agent will check its epistemic uncertainty in the current state (line 4) and, in case it is high[3] and the demonstrator is available (line 5), the agent will ask for an advice and follow the suggested action (line 6). Otherwise, the usual exploration will be applied (line 8). $\hat{Q}$ and π are updated normally

is possible to reduce by collecting more samples, while the latter is not.

[3]We decide if the uncertainty is high through a predefined threshold in our evaluations.

according to the chosen learning algorithm.

Algorithm 5 RCMP

Require: Value function approximator $\hat{Q}$, agent policy π, uncertainty estimator μ, demonstrator π_Δ, availability function $\boldsymbol{A}_t$

```
Initialize Q̂ and π
for ∀ learning step t do
    Observe s
    u ← μ(s)                          ▷ Calculate uncertainty (Eq. (4.5))
    if u is high and A_t(t) then
        a ← π_Δ(s)                    ▷ Ask for advice
    else
        a ← π(s)                      ▷ Normal policy
    Apply action a and observe s', r
    Update Q̂ and π with ⟨s, a, s', r⟩
```

Calculating the Uncertainty

Value-based algorithms estimate the expected value of applying each action in a given state. However, vanilla algorithms cannot estimate the uncertainty on their predictions, which means that we can compare the expected values of each action but there is no direct way of estimating the uncertainty of the predictions. For that purpose, we propose a way to measure the uncertainty with a small enhancement in standard algorithms. Consider the illustration of a DQN network in Figure 4.5a. The first layer consists of the state features, whereas the last layer outputs an estimate of the expected value for each action. We propose to add as a last layer multiple *heads* estimating separately expected values for each action, as done in Bootstrapped DQN (OSBAND et al., 2016). Due to the aleatoric nature of the exploration and network initialization, each head will output a different estimate of the action values. As the learning algorithm updates the network weights, their predictions will get progressively closer to the real function, and consequently one close to the others as the variance of the predictions is reduced. Therefore, we use the variance of the predictions across the heads as an estimate of uncertainty for a given state:

$$\mu(s) = \frac{\sum_{\forall a \in A} var(\boldsymbol{Q}(s,a))}{|A|}, \tag{4.5}$$

where $\boldsymbol{Q}(s,a) = \begin{bmatrix} Q_1(s,a) \\ \vdots \\ Q_h(s,a) \end{bmatrix}$, $Q_i(s,a)$ is the Q-value given by head i for state s and action a, var is the variance, and h is the chosen number of heads. The final value

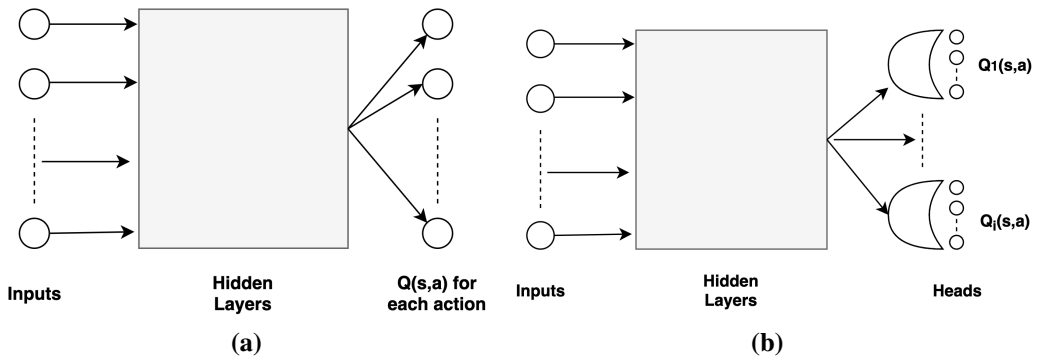


Figure 4.5. (a) Illustration of a regular DQN network and (b) a network with *heads*. Each head estimates a value for each action.

prediction (used, for example, for extracting a policy from the value function) is the average of the predictions given by each head:

$$\hat{Q}(s,a) = \frac{\sum_{i=1}^{h} Q_i(s,a)}{h}. \tag{4.6}$$

Each head will have their own loss function to minimize. The DQN algorithm might be adapted by calculating a loss for each head as:

$$L_i^{DQN} = \mathbb{E}_{\boldsymbol{D}}[(r + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a))^2], \tag{4.7}$$

where L_i^{DQN} is the loss function for head i and $\boldsymbol{D}$ is the minibatch for the update.

4.2.2 Implementation-friendly Description

Although the previous section fully describes our proposal, we show here further implementation details. Our description is a viable and efficient implementation of the training of DQN with multiple heads.

As illustrated in Figure 4.5b, we assume the network Q is implemented giving as output a prediction for each action $a \in A$ in each head $i \in \{1, \ldots, h\}$, given a batch of states $\boldsymbol{s}$. Therefore, the output of a forward pass in Q is of dimension $h \times |\boldsymbol{s}| \times |A|$. We assume that a *target* network Q^t is used for stability[4].

In practice, updating each head with different samples might be desired to help to reduce the bias that might artificially reduce the variance on the predictions. For this purpose, we make use of a *sample selecting function* $d : \boldsymbol{D} \times h \rightarrow \{0,1\}^{h \times |\boldsymbol{D}|}$, where $\boldsymbol{D}$ is the mini-batch for the current update. This function will sample either 0 (not

[4]In cases where no target network is used, simply consider $Q = Q^t$ in this section.

use) or 1 (use) for each sample and each head. The simplest way of implementing d is by sampling $|\boldsymbol{D}|h$ numbers from $\{0, 1\}$ with a fixed probability, but any other strategy might be used. Alternatively, different mini-batches could be sorted for each head, but using d is simpler to implement efficiently. Any network architecture might be used for the hidden layers according to the desired domain of application, as long as the input and output layers are defined as specified.

Algorithm 6 describes the implementation of the loss function for DQN, where vectors and matrices are in bold and the comments on the right side depict the dimensionality of the result of each calculation. For a particular minibatch $\boldsymbol{D}$, we first convert the applied actions to the one-hot representation (line 2). Then, we predict the values of the next states (line 3) and for the observed state-action tuples (line 4). Finally, we calculate a loss for each head, using the sorted samples (line 5) to calculate the predicted and target values (lines 7 and 8). Here, $\odot$ represents the element-wise multiplication.

Algorithm 6 Implementation of DQN with heads

Require: minibatch $\boldsymbol{D}$, Q-network Q, target network Q^t, sample selecting function d, number of heads h

1: $\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}, \boldsymbol{r} = \boldsymbol{D}$
2: $\boldsymbol{act} \leftarrow one\_hot(\boldsymbol{a})$ ▷ $|A| \times \boldsymbol{D}$
3: $\boldsymbol{q}^t \leftarrow \max_a Q^t(\boldsymbol{s'})$ ▷ $h \times |\boldsymbol{D}|$
4: $\boldsymbol{q} \leftarrow Q(\boldsymbol{s})\boldsymbol{act}$ ▷ $h \times |\boldsymbol{D}|$
5: $\boldsymbol{p} \sim d(\boldsymbol{D}, h)$ ▷ $h \times |\boldsymbol{D}|$
6: **for** $\forall i \in \{1, \ldots, h\}$ **do**
7: $\quad \boldsymbol{target} \leftarrow \boldsymbol{r} + \gamma \boldsymbol{q}^t[i] \odot \boldsymbol{p}[i]$ ▷ $|\boldsymbol{D}|$
8: $\quad \boldsymbol{pred} \leftarrow \boldsymbol{q}[i] \odot \boldsymbol{p}[i]$ ▷ $|\boldsymbol{D}|$
9: $\quad loss[i] \leftarrow \frac{1}{|\boldsymbol{D}|} \sum(\boldsymbol{target} - \boldsymbol{pred})^2$

4.2.3 Experimental Evaluation

We evaluated RCMP in two domains varying (i) learning algorithms; (ii) competency level of the demonstrators; (iii) domain complexity. The first one is a relatively simple gridworld-like domain where we can define the optimal policy and use it as a demonstrator for a DQN agent. The second one is the *Pong* Atari game, a much more complex domain where we use as demonstrator a previously-trained A3C agent (MNIH et al., 2016). With both evaluation domains we hope to show that RCMP is useful across different scenarios.

- **RCMP**: Our proposal as described in the last section.

- **No Advice**: A baseline learning with no advice.
- **Random**: The agent receives uniformly random advice with no regard to its uncertainty.
- **Importance**: Advice is given according to the importance advising metric calculated from the Q-function of a trained agent (Eq. (3.2)), as in previous action advising literature (TAYLOR et al., 2014b; AMIR et al., 2016). Notice that this algorithm is more restrictive than RCMP because: (i) the demonstrator needs a Q-function; (ii) the learning agents needs to be observed at all time steps, while for RCMP the own agent monitors its uncertainty and queries the demonstrator only when needed.

Gridworld

In the *Gridworld* domain, the agent aims at reaching a goal as quickly as possible, while avoiding falling in one of the holes spread in the environment. The agent has 4 actions $A = \{up, down, left, right\}$ that most of the times have the intended effect, unless the agent is in the 4-neighborhood of a hole. In that case, the agent has a 25% probability of falling into the hole regardless of the applied action. An episode ends when the agent has either reached the goal or fallen into a hole. In the former case, a reward of $+1$ is awarded, whereas in the latter the reward is -1.

Before evaluating the learning performance of the algorithms, we use this domain for analyzing the effect of the number of heads h on the uncertainty estimate. Figure 4.6 shows the average uncertainty observed in each learning episode over time for different configurations of the parameters. Regardless of the chosen parameter value the estimate works as expected. At first the uncertainty is high, then it gets progressively lower as the agent trains for longer. However, if the number of heads is very low ($h = 2$), sudden spikes in the uncertainty might be observed when the agent encounters new situations (e.g., around after 120 and 200 learning episodes). The uncertainty curve tends to become smoother for higher number of heads, as shown in the smooth curve of $h = 100$. However, adding more heads means adding parameters to be trained for each head, hence a trade-off is desired.

For evaluating the learning performance in this domain, we use DQN as the base learning algorithm and the optimal policy as the demonstrator. All algorithms are trained for 1000 episodes in total, where the agents are evaluated (exploration and updates turned off) for 10 episodes at every 10 learning episodes. The maximum number of demonstrated steps is set to 700 for the algorithms that can receive advice.For all

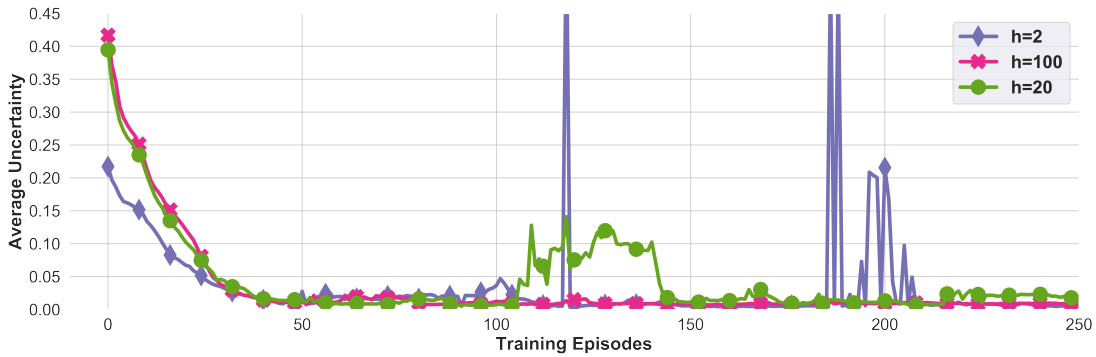
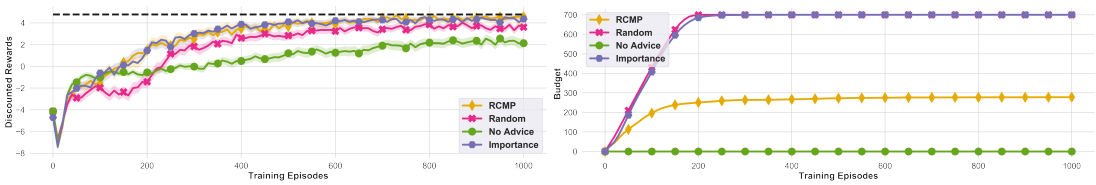


Figure 4.6. Average uncertainty of the learning episodes over time. Averaged over 200 repetitions.

algorithms, $\alpha = 0.01, h = 5$, and $\gamma = 0.9$. The network architecture is composed of 2 fully-connected hidden layers of 25 neurons each before the layer with the heads.



(a) **(b)**

Figure 4.7. (a) Discounted rewards and (b) amount of advice used in 200 repetitions of the *Gridworld* experiment. The shaded area corresponds to the 90% confidence interval. The dashed line corresponds to the optimal performance.

Figure 4.7a shows the performance in observed discounted reward for each algorithm, while Figure 4.7b shows the amount of advice used. RCMP asks for advice until around 200 learning steps, after which the algorithm already has high confidence on its predictions and stop asking for advice. Both *Random* and *Importance*, on their turn, keep asking for advice until the maximum budget is used. RCMP achieves better performance than both *Random* and *No Advice* and the ties with *Importance* for the best performance, while using less advice among all the advice-based algorithms. Notice that RCMP does not use the maximum budget, stopping to ask for advice when it is not expected to be useful anymore, while *Importance* and *Random* spend all the available advice regardless of how fast the learning agent converges. In all cases, the use of advice helped converging faster towards the optimal policy than *No Advice*. After 1000 episodes, *No Advice* still has not converged to the same performance as the algorithms making use of advice.

Figure 4.8 shows the accumulated reward achieved by each algorithm throughout the entire evaluation. In this experiment, RCMP performed better than both *No Advice* and *Random* and tied for the best performance with *Importance*, while using less

advice than all other advice-based algorithms.

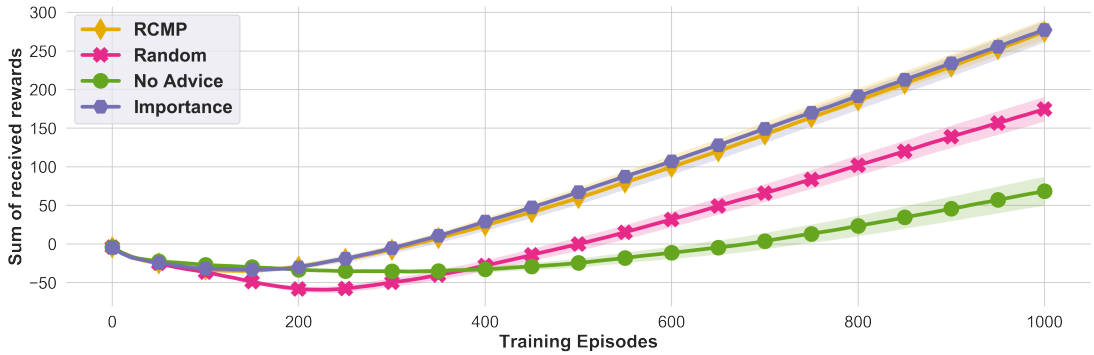
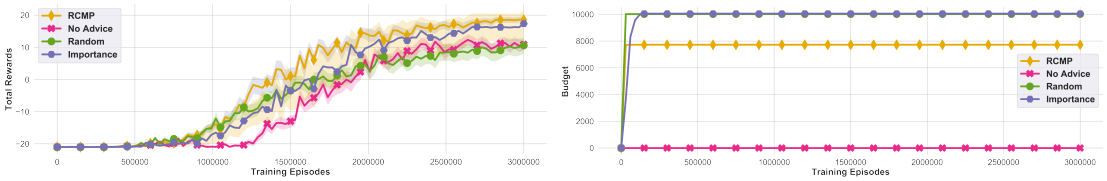


Figure 4.8. Sum of discounted rewards observed in 200 repetitions of the *Gridworld* experiment. The shaded area corresponds to the 90% confidence interval.

Pong

For this domain, we use A3C as the base learning algorithm. We train an A3C agent until it is able to achieve a score of +21 in an episode and use it as the demonstrator. All algorithms are trained for 3 million steps, where an evaluation phase of 1 episode is carried out after each $30,000$ learning steps. For all algorithms, $\alpha = 0.0001, h = 5$, and $\gamma = 0.99$. The network architecture is composed of 4 sequences of Convolutional layers followed by max pooling layers, connected to the critic head and actor layers that are fully-connected.

Figure 4.9a and 4.9b show, respectively, the undiscounted reward achieved by each algorithm and the amount of received advice. RCMP starts to show performance improvements over *No Advice* roughly around after $1,000,000$ learning steps. The apparent disconnect between when the agents receive advice and when the improvement happens is because a sequence of actions must be learned before an improvement in score is seen. Although all advice-based algorithms are getting closer to a winning behavior as they receive advice, seeing an improvement in score takes longer. While *Random* and *Importance* quickly spends all the available advice, RCMP asks for advice only for a short period of time, after which the uncertainty is not high enough to ask for it anymore. Although the pattern in advice use and improvement over *No Advice* is the same as for the *Gridworld* domain for all algorithms, here RCMP presents clear improvements over all the other algorithms while receiving less advice (more visible in Figure 4.10).



(a) **(b)**

Figure 4.9. (a) Undiscounted rewards and (b) amount of advice used in 20 repetitions of the *Pong* experiment. The shaded area corresponds to the 60% confidence interval.

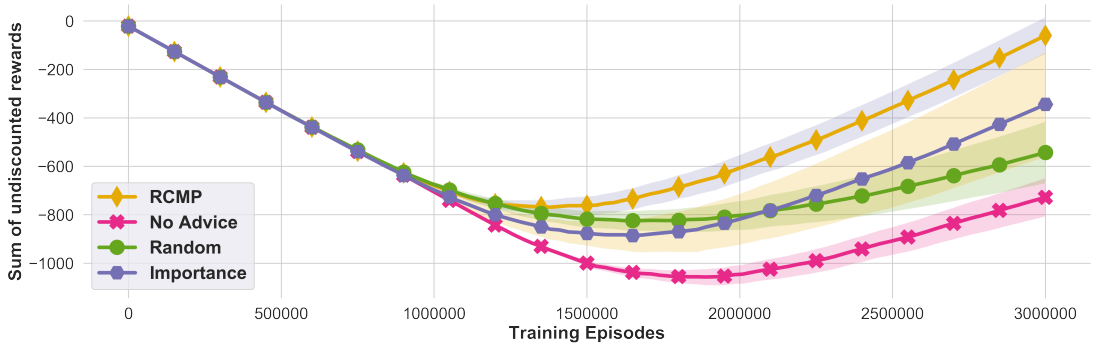


Figure 4.10. Sum of undiscounted rewards observed in 20 repetitions of the *Pong* experiment.

4.2.4 Discussion

We have evaluated RCMP in the *Gridworld* and *Pong* domains. The main conclusions drawn from our evaluation are:

- RCMP performs better than regular learning, randomly receiving advice, and importance advising across domains of different complexity levels.
- Receiving advice based on epistemic uncertainty is advantageous both when the demonstrator is optimal (*Gridworld*) or a trained agent with no optimality guaranteed (*Pong*).
- Our procedure to estimate epistemic uncertainty is effective and easily adaptable across different value-function-based learning algorithms (DQN and A3C were evaluated in this paper).

RCMP can now be combined with Ad Hoc Advising and evaluated in multiagent domains composed of Deep RL agents in further work.

Chapter 5

Object-Oriented Representation for Transfer

In order to perform TL across multiples tasks or agents, following an abstracted representation might help the learning agents to generalize experiences and relate entities from different tasks. In this Chapter, we describe our advancements related to the Object-Oriented task description. We first extend the OO-MDP formulation to the multiagent case (Section 5.1 - previously published at (SILVA; GLATT; COSTA, 2016; SILVA; GLATT; COSTA, 2019)). Then, we propose a method for mapping entities and transferring knowledge across tasks based on the object-oriented representation (Section 5.2 - previously published at (SILVA; COSTA, 2017c)). Finally, we show how the object-oriented representation can be used to automatically generate tasks and organize them in a *Curriculum* (Section 5.3 - previously published at (SILVA; COSTA, 2017b; SILVA; COSTA, 2018)).

5.1 Object-Oriented Representation in MAS

While relational approaches have benefited MAS in previous work (CROONENBORGHS et al., 2005; PROPER; TADEPALLI, 2009), OO-MDP has not been applied in multiagent RL so far. To the best of our knowledge, up to now the only effort toward extending OO-MDP to MAS appears in BURLAP (MACGLASHAN, 2015), a library for the development of planning and learning algorithms based on OO-MDP. However, there is no formal OO-MDP framework for MAS nor distributed solutions based on OO-MDP for a generic number of agents.

Inspired by the insight that each agent in a MAS can be seen as an object, we ex-

tend OO-MDP for MAS, defining the *Multiagent Object-Oriented MDP* (MOO-MDP). We also present a *model-free* algorithm based on Distributed Q-Learning (LAUER; RIEDMILLER, 2000), hereafter called *Distributed Object-Oriented Q-Learning* (DOO-Q), that can solve deterministic distributed MOO-MDPs in cooperative domains in which all agents try to maximize the same reward function (PANAIT; LUKE, 2005). DOO-Q reasons over abstract states, which help to accelerate learning. We also show that, under certain constraints, DOO-Q does not need to consider all the concrete state space and still learns optimal policies. Thus, in summary, this section answers the following questions: (i) How to describe a multiagent RL domain with multiple autonomous agents in an object-oriented manner, and (ii) How to learn, in a distributed manner, an optimal *joint* policy in deterministic cooperative MOO-MDPs.

Here, we present a formal definition for an MOO-MDP, an extension of OO-MDP to MAS. MOO-MDP follows the premises of full observability. We are interested in a distributed control, in which agents cannot tell other agent's actions.

The main differences between OO-MDPs and MOO-MDPs are that in the latter:

1. the environment is simultaneously affected by multiple agents, which means that the state transition now depends on *joint* actions rather than individual agent actions;

2. each agent may have a slightly different observation of the world, resulting in similar but possibly different local states; and

3. each agent may have its own reward function, which means that agents can have different goals.

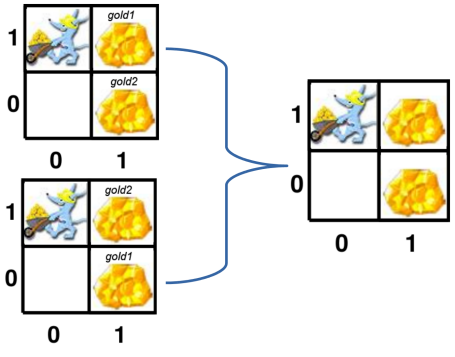
We focus on cooperative domains here, however MOO-MDP is a general model that can be used for both cooperative and competitive MAS (or a mix of those).

An MOO-MDP is described by the tuple $\left\langle C, O, U, \mathrm{T}, D, R^M \right\rangle$. C is the set of *classes* and m is the number of agents. We define $Ag = \{Z_1, \ldots, Z_g\}$, $Ag \subseteq C$ as the set of *Agent Classes*, i.e., each object of any class $Z_i \in Ag$ is an agent able to observe the environment and perform autonomous actions. Note that $g \leq m$, because more than one autonomous agent may belong to the same class. $\Gamma \subseteq C$ is the set of *abstracted* classes. Objects of these classes cannot be differentiated among themselves except by their attribute values, thus each object assumes an *abstract* state $o.\overline{state} = \prod_{b \in Att(C(o))} o.b$. Notice that *abstract* states suppress object ids, as different objects may be taken as equivalent as long as they have the same attribute values and belong to the same class.

O is the set of objects that is divided as $O = E \cup G$, where E is the set of environment objects (not related to agents), $\forall e \in E : C(e) \notin Ag$, and G is the set of agent objects, $\forall z \in G : C(z) \in Ag$. The current *concrete* state now is a composition of the state of environment and agent objects $s = \bigcup_{o \in O} o.state = (\bigcup_{e \in E} e.state) \cup (\bigcup_{z \in G} z.state)$, meaning that, if the state of all agents cannot be directly observed, these states must be received through communication in all decision steps. An abstract state $\tilde{s}$ is defined according to $\tilde{s} = \left(\bigcup_{o \in O, C(o) \in \Gamma} o.\overline{state}\right) \cup \left(\bigcup_{o \in O, C(o) \notin \Gamma} o.state\right)$, where $\tilde{s}$ is a set of concrete states ($\tilde{s} \subseteq S$). The function κ defines the abstract state for an agent z given a concrete state s: $\tilde{s}^z = \kappa(s, z)$, in which all objects of classes $C_i \in \Gamma$ have their id suppressed. Note that the definition of abstract states enables knowledge generalization. For example, in the *Goldmine* domain we set the *Gold* class as abstracted, $Gold \in \Gamma$, so that when an agent first collects a gold piece, it will learn that this action results in a positive reward and, as *Gold* is abstracted, the agent generalizes this knowledge and reasons that any collected gold piece results in a positive reward. Figure 5.1 further illustrates how the abstraction works in a 2×2 *Goldmine* domain. Note that multiple concrete states are compressed into a single abstracted one.

U is the set of joint actions for all agents, in which each agent z has its own individual action set A^z, $U = A^1 \times \cdots \times A^m$. Individual actions can be parameterized or not, thus MOO-MDPs also allow action space abstraction. Agent action sets do not need to be equal, hence MOO-MDPs can be applied in both homogeneous and heterogeneous MAS.

Both ϱ and D have the same definition as in OO-MDPs, and $R^M = \{R^1, \ldots, R^m\}$



(a)

$s_1 =$	$\{miner = \{0,1\}, gold1 = \{1,1\}, gold2 = \{1,0\}\}$
$s_2 =$	$\{miner = \{0,1\}, gold1 = \{1,0\}, gold2 = \{1,1\}\}$
$\tilde{s} =$	$\{miner = \{0,1\}, gold = \{1,0\}, gold = \{1,1\}\}$
$\tilde{s} =$	$\{s_1, s_2\}$

(b)

Figure 5.1. (a) Graphical representation and (b) textual representation of the state space abstraction. s_1 and s_2 represent two concrete states (ids on top of gold pieces) that are described by a single abstract state $\tilde{s}$ in the right side when $Gold \in \Gamma$.

is the set of reward functions for all agents, which now returns reward signals taking *joint* actions into account, rather than individual actions. While A^z is known, ϱ, D, R^M, and U are unknown to the agent in a learning task.

The transition dynamics is fully described in Algorithm 7.

In each step k, all agents observe their abstract state $\tilde{s}_k^z$ and apply an action $a_k^z \in A_z$. All terms are evaluated according to s_k (defined from O_k) and the joint action $\boldsymbol{u}_k$ triggers all effects related to matched conditions in the rules $d \in D$. Finally, effects change object attributes, causing a state transition, and the agent receives a reward r_k^z. Note that state transitions depend on both *term* values (defined over *concrete* states) and the *joint* action. As the set of *terms* and concrete states are unknown to the agent for learning problems, it has to reason over only abstract observations of the environment.

In the next section we present a model-free algorithm to solve deterministic cooperative MOO-MDPs with homogeneous agents.

5.1.1 Learning in deterministic cooperative MOO-MDPs

We are interested in a solution for Deterministic Distributed Cooperative MOO-MDPs, a specific class of the general MOO-MDP framework presented in the previous section. The reward function in cooperative MAS is the same for all agents, $R^1 = \cdots = R^m$. We assume that there is no central controller, and each agent takes actions unaware of other agent actions. We here present a *model-free* algorithm based on Distributed Q-Learning (LAUER; RIEDMILLER, 2000) to solve such problems.

The proposed algorithm, thereafter called *Distributed Object-Oriented Q-Learning* (DOO-Q), learns a joint policy in a distributed and generalized manner, in which each agent z stores a Q-table (Q^z) containing abstract states ($\tilde{s}_k^z$) and its own actions. DOO-Q reasons over concrete actions, which means that all parameterized actions must be

Algorithm 7 Environment dynamics for MOO-MDPs

Require: complete MOO-MDP description, the initial state s_0 of objects $o \in O_0$.

1: **for** Each step $k \geq 0$ **do**
2: All agents $z \in G$ observe state $\tilde{s}_k^z = \kappa(s_k, z)$.
3: Let all agents perform their actions.
4: Assemble joint action $\boldsymbol{u_k} = [a_k^1, \ldots, a_k^m]^T$.
5: Evaluate all *terms* $t \in T$.
6: Define all triggered *effects* in *rules* $d \in D$.
7: Define state s_{k+1} according to triggered effects.
8: Define rewards R_k^M and return $(r_k^z, \tilde{s}_{k+1}^z)$ to all agents.

grounded before execution. We leave the action space abstraction for further works.

An agent z can find a suitable policy for the joint actuation, even when unaware of other agent actions, through iteratively updating its Q-table over abstract states and concrete actions:

$$Q^z_{k+1}(\tilde{s}^z_k, a^z_k) \leftarrow \max\{Q^z_k(\tilde{s}^z_k, a^z_k), r_k + \gamma \max_{a^z \in A_z} Q^z_k(\tilde{s}^z_{k+1}, a^z)\}. \tag{5.1}$$

Lauer and Riedmiller (LAUER; RIEDMILLER, 2000) proved that, when using only concrete states, this update-rule allows agents to learn a projection of the *joint* Q-table in a distributed manner. We apply Equation (5.1) with abstract states in order to learn an optimal joint policy while storing only a local Q-table, which can be done because it corresponds to the joint Q-table as

$$Q^z_k(\tilde{s}^z_k, a^z_k) \geq \max_{\boldsymbol{u} \in U, \boldsymbol{u}^z = a^z_k, s \in \tilde{s}^z_k} Q_k(s, \boldsymbol{u}), \tag{5.2}$$

where Q^z_k is the local Q-table of agent z at step k, Q_k is the joint Q-table for all agents, where agent z chose action a^z_k ($\boldsymbol{u}^z = a^z$), and $s \in \tilde{s}^z_k$. Local Q-values are defined for a given abstract state $\tilde{s}^z_k$ and an agent action $a^z_k \in A_z$, while joint Q-values are defined for concrete states s and joint actions $\boldsymbol{u} \in U$, composed of actions of all agents. During the learning process, a single value of the local Q-table can be greater than the joint Q-table values, because another concrete state $s' \in \tilde{s}^z_k$ may have been visited before, a situation in which generalization causes a faster convergence. For a proof that that Equation (5.2) holds for MOO-MDP when using abstract states, refer to Appendix B.

However, the greedy policy applied to local Q-tables is not guaranteed to result in an optimal joint policy because agents do not take into account each others actions and miss-coordination issues may arise. This means that agents need an additional coordination method for optimal actuation. Thus, each agent only updates its policy when a new action results in an improvement over all other actions previously applied in the current state. This update-rule solves coordination issues since all agent policies will repeat the first joint action that received the optimal discounted reward and is described by

$$\pi^z_{k+1}(\tilde{s}^z) \leftarrow \begin{cases} \pi^z_k(\tilde{s}^z) & \text{if } \max_{a^z \in A_z} Q^z_k(\tilde{s}^z, a^z) = \max_{a^z \in A_z} Q^z_{k+1}(\tilde{s}^z, a^z) \\ a^z_k & \text{otherwise} \end{cases}. \tag{5.3}$$

As a greedy policy applied to a joint Q-table in cooperative scenarios leads to an optimal actuation, a distributed policy is optimal if it is greedy with respect to the joint Q-table. We prove that Equation (5.3) has this property in Appendix C.

DOO-Q solves MOO-MDPs allying distributed Q-table update of Equation (5.1) with policy update of Equation (5.3). DOO-Q is fully described in Algorithm 8. At first, local Q-tables are initialized with zero values according to Assumption 2 of Proposition 1. Then, each agent observes its current abstract state $\tilde{s}_k^z$ according to the state of all objects, and chooses an action a_k^z according to its exploration strategy $ExpStr$. Any function that has a non-zero probability of executing all applicable actions can be used as $ExpStr$ (as required by Proposition 2), e.g. the ϵ-greedy strategy. The $ExpStr$ arguments are $\tilde{s}_k^z$, to know which actions are applicable, and the current policy π_k^z. After all agents have applied their actions, each agent observes its next state and reward, and finally updates its Q-table and policy π_k^z, ending the current learning step.

Note that the agent's observation of the current state over the set of objects enables state generalization; i.e. an agent may see all objects of a given class as equivalent, and only differentiate them by their attribute values. Note also, that here the environment returns a single reward r_k to all agents.

Algorithm 8 Learning for a DOO-Q agent z

Require: exploration strategy $ExpStr$, discount rate γ, abstraction function κ, state space S, and action space A^z.

1: $Q_0^z(\kappa(s,z),a) \leftarrow 0, \forall s \in S, a \in A^z$.
2: Initiate π_0^z as a greedy policy.
3: Observe current abstract state $\tilde{s}_0^z$.
4: **for** Each learning step $k \geq 0$ **do**
5: Apply action $a_k^z = ExpStr(\tilde{s}_k^z, \pi_k^z)$
6: Observe reward r_k and new state $\tilde{s}_{k+1}^z$.
7: Update $Q_k^z(\tilde{s}_k^z, a_k^z)$ (Equation 5.1).
8: Update policy $\pi_k^z(\tilde{s}_k^z)$ (Equation 5.3).
9: $\tilde{s}_k^z \leftarrow \tilde{s}_{k+1}^z$.

In the next Section we present an experimental evaluation to compare DOO-Q with other algorithms to solve cooperative SG.

5.1.2 Experimental Evaluation

We evaluate DOO-Q in three domains. The first evaluation, in the *Goldmine* domain, is designed to represent situations where the object-oriented representation benefits from domain characteristics, while all the assumptions of our theoretical proofs hold. The second one, in the *Gridworld* domain, is designed to be a simple domain with small state space, in which the task is easy to solve without abstraction. While the performance of algorithms that reason over concrete states is maintained, our proposal

has better memory requirements in such domains. The third one, in the *Predator-Prey* domain, we solve a task with partial observability and a non-deterministic environment in which some of the assumptions of our theoretical proof for learning an optimal policy are violated. This last domain shows experimental evidence of the robustness of our proposal under conditions not covered by our theoretical analysis.

For all domains, the performance of the following algorithms were compared:

- **Single-agent Q-Learning (SAQL)**: We adapt the single-agent Q-Learning (WATKINS; DAYAN, 1992) to MAS. A central controller is designated to control all agents in the environment. A joint state-action space is used to build the Q-table. Here, the Q-table update is computed with *joint* actions. The Object-Oriented representation is used to define the state space.

- **Multiagent Q-Learning (MAQL)**: Each miner is an autonomous agent in this algorithm. Agents cannot communicate, but they are able to observe each others actions in all steps. Thus, each agent stores a Q-table that has an entry for all states and *joint* actions, and every agent actuates believing that all other agents will choose the individual action which has the maximum Q-value.

- **Distributed Q-Learning (DQL)**: The standard Distributed Q-Learning (LAUER; RIEDMILLER, 2000) is similar to our proposal, but without using the Object-Oriented representation (i.e., it does not allow state abstraction).

- **DOO-Q**: In our proposal, each agent is autonomous and selects a local action based on local abstract states.

In all domains and for all algorithms we use the ϵ-greedy exploration strategy with $\epsilon = 0.1$. In the following Sections we describe the results in each evaluation domain. The algorithms are compared based on Q-table size and learning speed. While the former can be analyzed theoretically, the latter is evaluated through experiments.

The number of Q-table entries for an algorithm depends on the size of the state and action spaces, $|Q| = |S| \times |A|$. For both the *Goldmine* and *Gridworld* domains we present a theoretical definition of the Q-table entries that are necessary for each algorithm, and compare the performance of each of them through the cumulative reward achieved in the evaluation episodes. For the *Predator-Prey* domain we show the number of used Q-table entries and the performance of each algorithm (the average number of steps to solve the evaluation episodes). In the next sections we present the achieved results.

5.1.3 Goldmine Results

In our experiments, we randomly generated 70 initial states in a 5×5 grid with 3 miners and 6 gold pieces (Figure 2.1 is an example of such states) and used them to compare the performance achieved by each of the algorithms. The experiment was designed in a way that every algorithm experiences the same initial states in the same order, and the next initial state is defined by swapping the position of objects of the same class after each episode. For each of the states, algorithms explore using an exploration strategy and, after every interval of 100 episodes, a single episode is assessed using the greedy policy to extract the number of steps required to reach a terminal state and the received accumulated discounted reward. The algorithms were configured as follows:

1. **SAQL**: This algorithm was used in the original *Goldmine* modeling, where an external agent sees each miner as a simple environment object (and not as an autonomous agent). A single miner can be moved at each step and all decisions are made by the external agent, which means miners do not perform actions by themselves. The Q-Learning algorithm was used to solve the task, with the parameters $\alpha = 0.2$, $\gamma = 0.9$. Here, $\Gamma = \{Gold, Wall\}$. We used the default implementation available in the BURLAP framework (MACGLASHAN, 2015).

2. **MAQL**: The following parameters were set: $\alpha = 0.2$, $\gamma = 0.9$, and $\Gamma = \{Miner, Gold, Wall\}$. The BURLAP default implementation was used.

3. **DQL**: This algorithm is implemented with a factored state description and $\gamma = 0.9$.

4. **DOO-Q**: For our proposal, $\gamma = 0.9$ and $\Gamma = \{Miner, Gold, Wall\}$.

A time limit was set for the experiment, in which an algorithm is interrupted if it takes too long to conclude a predefined number of learning episodes. In this case, the results achieved so far were still stored.

We firstly present the number of Q-table entries for each algorithm. Let m be the number of miners, p be the number of gold pieces, q be the number of individual actions affecting a single miner state, and w be the number of possible cells inside the grid. In order to simplify calculations, we assume that Q-table entries are created even if actions are not applicable in a given state.

- **SAQL**: One agent controls all miners, but only moves one single miner per step, so we get the size of the action space $|A| = q\,m$. The state space is defined over all

possible grid cells that each miner and each gold piece can occupy (gold pieces can also be in *collected* state). As *Gold* is an abstracted class, the number of ways that gold pieces can be dispersed in the grid is calculated as a permutation with repetitions, leading to $|S| = w^m \frac{(p+w)!}{p!w!}$. The memory requirement is then $O\left(w^m \frac{(p+w)!}{p!w!} q\ m\right)$.

- **MAQL**: One agent for each miner, which move simultaneously in every step, thus all possible combinations of joint actions determine the size of the action space $|A| = q^m$. As *Miner* and *Gold* are abstracted classes, only the agent is distinguishable by the id, resulting in $|S| = w \frac{(m+w-2)!}{(m-1)!(w-1)!} \frac{(p+w)!}{p!w!}$. The memory requirement for this algorithm is $O\left(w \frac{(m+w-2)!}{(m-1)!(w-1)!} \frac{(p+w)!}{p!w!} q^m\right)$.

- **DQL**: Here, each agent only considers its actions leading to $|A| = q$. However, no abstraction is used, leading to $|S| = w^m(w+1)^p$. Thus, the memory requirement for DQL is $O(w^m(w+1)^p q)$.

- **DOO-Q**: The state space is the same as in MAQL and the action space is the same as in DQL. Thus, the memory requirement for DOO-Q is $O\left(w \frac{(m+w-2)!}{(m-1)!(w-1)!} \frac{(p+w)!}{p!w!} q\right)$.

For example, in a 5×5 environment with three miners, six gold pieces and fixed walls (as in our experiment), the number of Q-table entries per agent for each algorithm is roughly (i) **SAQL**: 1.7×10^{11}, (ii) **MAQL**: 7.4×10^{11}, (iii) **DQL** 2.4×10^{13}, (iv) **DOO-Q** 2.9×10^{10}.

Figure 5.2 depicts the results of the *Goldmine* experiment described in Section 5.2.2 in terms of discounted cumulative reward, and Figure 5.3 shows the number of steps taken until a terminal state is reached.

Figure 5.2 shows that DOO-Q learns an effective policy much faster and achieves higher rewards than all other algorithms since the beginning, maintaining better results until the end of the experiment. MAQL started with a performance comparable to SAQL, however the high memory usage made MAQL slower to process and the time limit was exceeded after only 700 learning episodes. When compared to the object-oriented algorithms, DQL presented a very slow learning process until the end of training. As the only difference between DOO-Q and DQL is the object-oriented representation, the results clearly reflect the advantage of MOO-MDPs in environments similar to *Goldmine*. Figure 5.3 shows that the MAS approaches (DOO-Q, MAQL and DQL) completed the task with less steps in the beginning of the training. Then, DOO-Q learned how to complete the task with very few steps after 1300 learning episodes and SAQL surpassed DQL after around 2000 episodes, because DQL presented a very slow learning. MAQL would probably present better results than SAQL in steps for

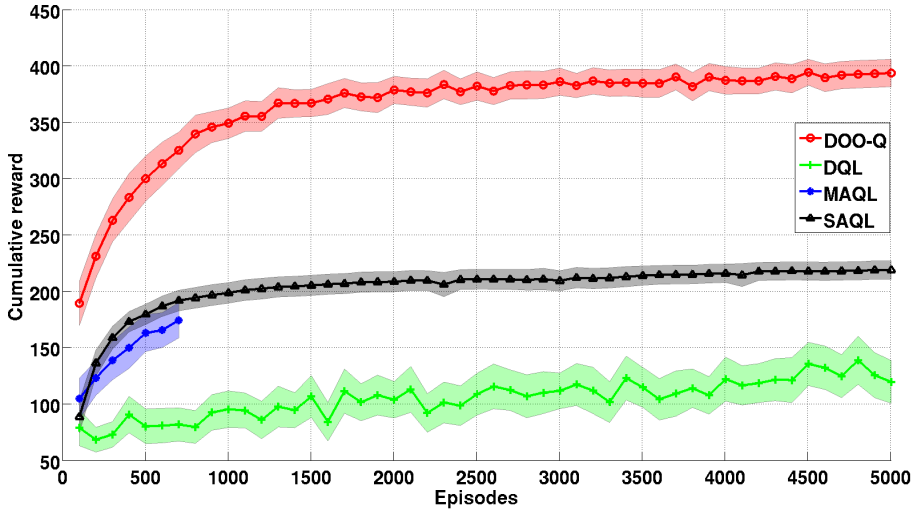


Figure 5.2. Observed discounted cumulative reward in the *Goldmine* domain. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The vertical axis represents the metric evaluated every 100 episodes of exploration. The shaded area represents the 95% confidence interval observed in 70 repetitions.

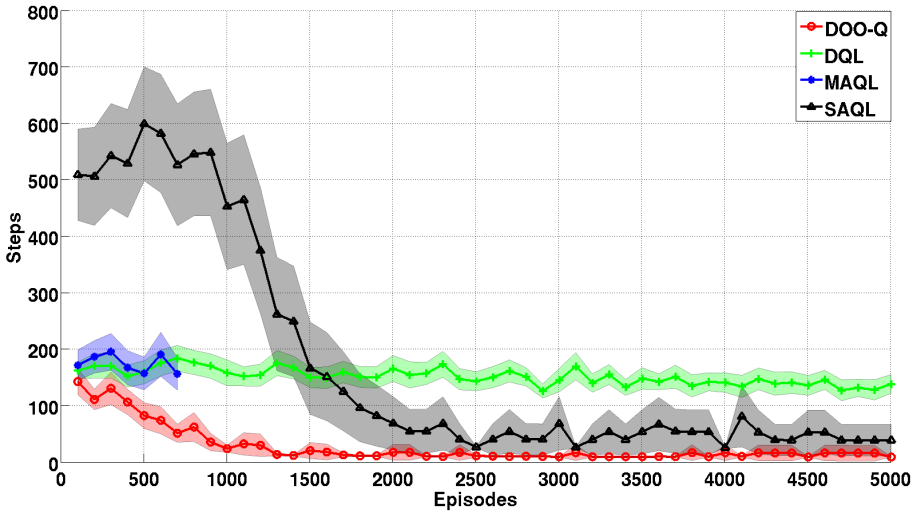


Figure 5.3. Observed number of steps to complete one episode in the *Goldmine* domain. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The vertical axis represents the metric evaluated every 100 episodes of exploration. The shaded area represents the 95% confidence interval observed in 70 repetitions.

task completion but it was unable to scale to this problem size because of computational limitations. The results in this domain show that, when applicable, abstraction greatly accelerates the learning speed, as DOO-Q achieved much better results than DQL. Also, compared to SAQL, MAS algorithms were able to learn to reduce the number of steps needed to complete the task faster, which indicates that dividing the workload helps to solve some problems.

Thus, DOO-Q achieved the best performance of the evaluated algorithms by using

the least space for the Q-table and by learning a good policy for a higher discounted cumulative reward much faster in the *Goldmine* domain.

5.1.4 Gridworld Results

For all algorithms, $\alpha = 0.1$ and $\gamma = 0.9$. Only the Multiagent approaches were evaluated in this domain (MAQL, DQL, and DOO-Q). For both DOO-Q and MAQL, we chose $\Gamma = \{Agent, Goal, Wall\}$.

The discounted cumulative reward achieved through exploiting the current learned policy was evaluated after every two episodes of learning, until 150 learning episodes were completed. The whole experiment was repeated 100 times to achieve statistical significance.

Let m be the number of agents, w be the number of possible positions, and q be the number of possible actions. For a *Gridworld* with fixed goals and walls the following memory requirements are demanded by each algorithm:

- **MAQL**: A Q-table entry is created for all possible combinations of joint actions, hence the size of the action space is $|A| = q^m$. As *Agent* is an abstracted class, and only one agent can be at a given position at a time step, the state space size is $|S| = w\frac{(w-1)!}{(w-m)!(m-1)!}$. The memory requirement for this algorithm is $O\left(w\frac{(w-1)!}{(w-m)!(m-1)!}q^m\right)$.
- **DQL**: Each agent only considers its actions leading to $|A| = q$. However, agents are not abstracted, leading to $|S| = \frac{w!}{(w-m)!}$. Thus, the memory requirement for DQL is $O\left(\frac{w!}{(w-m)!}q\right)$.
- **DOO-Q**: The state space is the same as in MAQL, however each agent only considers its actions leading to $|A| = q$. In this case the memory requirement for DOO-Q is $O\left(w\frac{(w-1)!}{(w-m)!(m-1)!}q\right)$.

For example, in a 4×3 grid with three agents, the number of Q-table entries per algorithm is roughly (i) **MAQL**: 8.25×10^4, (i) **DQL** 6.6×10^3, (iii) **DOO-Q** 3.3×10^3.

Figure 5.4 shows the difference between the achieved discounted cumulative rewards for each algorithm. Although DOO-Q and DQL Q-table sizes are smaller than MAQL, in practice only a small part of the state space is explored at first because the initial state is always the same. In this situation, MAQL learns how to coordinate faster because the other agent actions are observable. However, after roughly 25 episodes,

MAQL got stuck in a suboptimal policy, while DOO-Q and DQL remained improving their policies until the experiment was over. MAQL takes longer to improve its policy after episode 25 because of its large Q-table size, which renders the exploration less effective and prone to take a large amount of time to converge. As expected, the difference between DOO-Q and DQL in terms of accumulated reward is not statistically significant. However the memory requirements for DOO-Q are smaller than for all other algorithms.

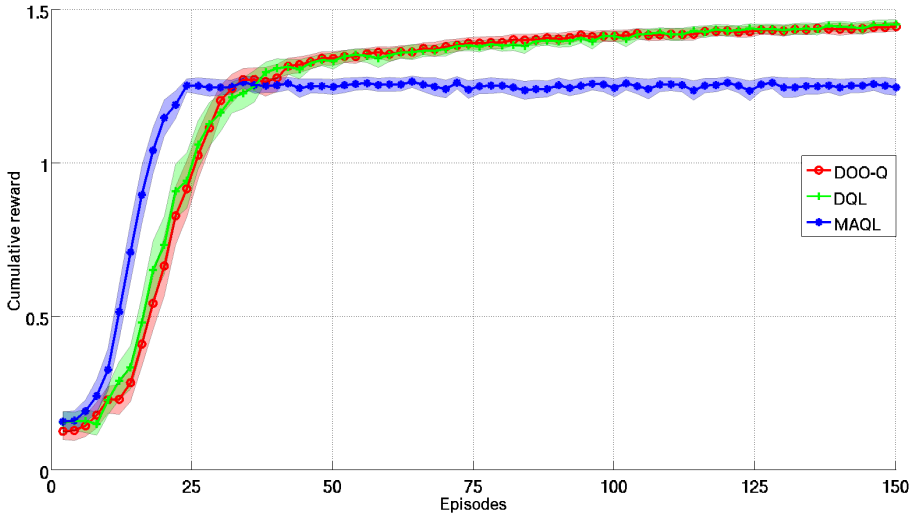


Figure 5.4. Observed discounted cumulative reward in the *Gridworld* domain. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The vertical axis represents the distributed accumulated reward evaluated every two episodes of exploration. The shaded area represents the 95% confidence interval observed in 100 repetitions.

5.1.5 Predator-Prey Results

Notice, that because of the random movements executed by the prey, the state transition function is non-deterministic, which means that Assumption 1 and 4 of Proposition 1 do not hold for this domain. However, even though the convergence to an optimal policy is not guaranteed, we show that our proposal learns how to solve the task. Here, *SAQL* can move all agents at each time step, which means that each Q-table entry contains the observations of all agents and a *joint* action. Furthermore, we chose $\Gamma = \{Prey, Predator\}$, $\gamma = 0.9$ and $\alpha = 0.1$ for all algorithms. We show the results for a task with 3 predators configured with $depth = 3$ trying to catch 2 preys.

Figure 5.5 shows the achieved performance for all algorithms. After 200 learning episodes *DOO-Q* solves the task with 80 steps on average, while *DQL*, *MAQL*, and *SAQL* solve the same task in 90, 92, and 85 steps, which means that *DOO-Q* learns how to solve the task more efficiently than the other algorithms. For the rest of the

training process the performance achieved by *DOO-Q* is still better than of the other algorithms. While *SAQL* has a good performance at the beginning of training, after 200 episodes it improves its performance very slowly, which makes *DQL* become faster after roughly 600 learning episodes. *MAQL* is slower than all other algorithms since the start, and all algorithms are improving their policies only very slowly after 1500 learning steps. *DOO-Q* is significantly better than all other algorithms according to the Wilcoxon signed rank test with 99% of confidence since 200 learning episodes.

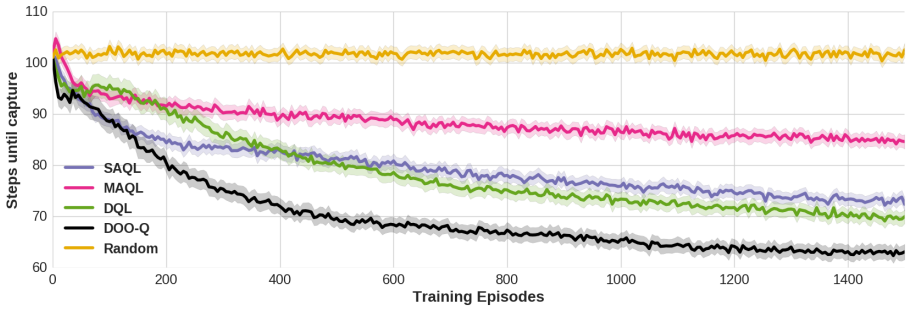


Figure 5.5. Observed average number of steps to capture the prey in evaluation episodes. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The shaded area represents the 99% confidence interval observed in 1000 repetitions. The performance achieved by a random agent is included as a baseline.

In addition to presenting a better performance, *DOO-Q* (together with *DQL*) uses much less memory than the other algorithms, as shown in Figure 5.6. While *DOO-Q* and *DQL* used less than 20 000 Q-table entries, *SAQL* and *MAQL* used roughly, respectively, 76 000 and 29 000 entries. The difference of memory usage between *DOO-Q* and *DQL* can be better visualized in Figure 5.7. While *DQL* used on average 15 500 Q-table entries after the learning process, *DOO-Q* used 12 000.

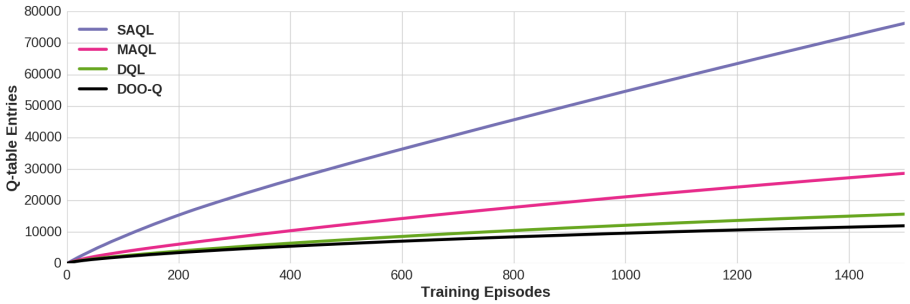


Figure 5.6. Observed Q-table size in the *Predator-Prey* domain. The horizontal axis is the number of executed episodes using the ϵ-*greedy* policy. The vertical axis represents the average Q-table size at that step.

In summary, our experiments show that DOO-Q achieves the best performance among the evaluated algorithms. While in the most favorable cases DOO-Q learned *faster* than the other algorithms with *fewer* memory requirements, in the most unfa-

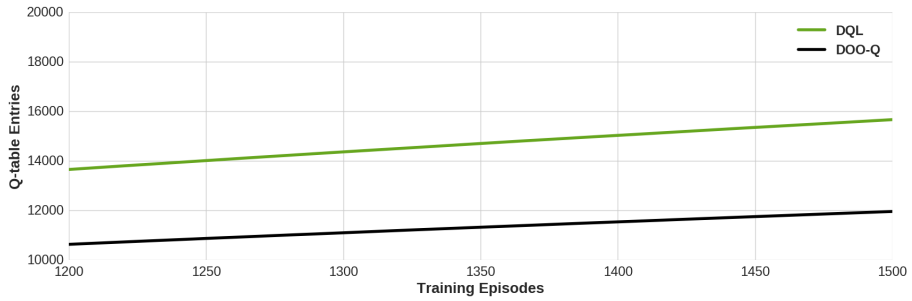


Figure 5.7. Amplification of Figure 5.6 to depict the difference between *DQL* and *DOO-Q*.

vorable case the performance was equivalent to the best algorithm (DQL) whereas the advantage of the reduced memory requirements remained steady.

5.2 Inter-Task Mapping

We are here interested in autonomously learning Inter-Task Mappings and transferring knowledge across tasks, but without collecting samples in the target task. Our proposal uses approximately the same domain knowledge given in *Mapping Learning via Classification* (TAYLOR; WHITESON; STONE, 2007), but with no samples in the target task.

We argue that estimating such a mapping is possible because the domain knowledge contained in an OO-MDP task description (MOO-MDP for MAS) is able to determine an initial estimate of how the source and target tasks are related. Since the state space is described by objects, an Inter-Task Mapping function needs to relate objects between tasks.

Taking into account the object-oriented representation, the description of source and target tasks may be different in regard to:

- $C^{source} \neq C^{target}$: The set of classes may be different in the source and target class. As the two tasks must have similarities in order to enable TL, we assume that a class mapping $\mathcal{X}_C$ is given, and that $\exists C_s \in C^{source} : \mathcal{X}_C(C_s) \in C^{target}$.

- $\exists C_s \in C^{source}, C_t \in C^{target}, C_t = \mathcal{X}_C(C_s)$: $Att(C_s) \neq Att(C_t)$: The attributes of one or more classes were changed (either attributes were removed or added).

- $\exists att_s \in Att(C^{source}), att_t \in Att(C^{target}), att_t = att_s$: $Dom(att_s) \neq Dom(att_t)$: The domain of one or more attributes has changed.

- $|O^{source}| \neq |O^{target}|$: The number of objects in the environment has changed.

Therefore, the TL algorithm must be able to cope with those kind of differences across tasks. For our TL proposal, as we don't have samples of the target task to precisely identify the relation between attributes of objects and the reward and transition functions, we define a Probabilistic Inter-TAsk Mapping (PITAM), $\mathscr{P}_o$, instead of a deterministic one:

$$\mathscr{P}_o = S^{target} \times S^{source} \rightarrow [0, 1]. \tag{5.4}$$

Remember that here the states are defined from sets of objects O^{target} and O^{source}. As we need $|O^{source}|$ objects in order to reconstruct a state in the source task, and $|O^{target}|$ may have a different value, a mapping $\mathscr{P}_o$ defines a probability of a state, i.e., a set of objects in the target task, $s_t = \bigcup_{o \in O^{target}} o.state$, being related to each possible set of objects in the source task, $s_s = \bigcup_{o \in O^{source}} o.state$, and this probability function may be used by the TL algorithm. We learn a mapping $\mathscr{P}_o$ by following Algorithm 9, named *Zero-Shot Autonomous Mapping Learning*. Here we consider that there is one source task and one target task. This procedure is called each time a new state in the target task is mapped. Firstly, the agent counts the number of objects of each class in the source and target task (n_s and n_t). Then, for each class C_i in the source task, the set P_{C_i} is built with all possible n_s-combinations of objects that belong to $\mathscr{X}_C(C_i)$ in the state s_t to be translated. After this process is executed for all classes in C^{source}, P contains sets of objects of all classes, and the $assemble()$ operation assembles states described with $|O^{source}|$ objects. This procedure is executed by considering all possible Cartesian products that contains one element of each $P_{C_i}, C_i \in C^{source}$. After the assemble operation is processed, we have Θ, $|\Theta| \geq 0$, which is the set of states in the source task for which a mapping from s_t is defined with a non-zero probability. The mapping probability is related to a similarity metric calculated for each state in Θ. This metric must reflect an estimate of how close a state in the source task is in regard to the state being mapped from the target task, and should take into account possible changes of class description and/or attribute domains. We consider here that all states that exist in the source task as equally similar to the desired target state:

$$similarity(s_t, s_s) = \begin{cases} 0 & \text{if } invalidState(s_s) \\ 1 & \text{otherwise} \end{cases}, \tag{5.5}$$

where $invalidState(O)$ is true if $\exists o \in s_s, a \in Att(C(o)) : o.a \notin Domain(a)$. The probability value is finally defined according to the relative similarity of each state. The intuition behind this similarity metric is that all valid mapped states, in which all attribute values for all objects belong to the interval defined by the attribute domain in the

source task, will be mapped with equal probability in $\mathscr{P}_O$. In case any attribute value falls outside the domain, the probability for that state is 0. In case a class definition in the target task has more attributes than in the source, the extra attributes are ignored. If the target task has fewer attributes, the ones that do not exist in the target task receive a valid standard value (*Unknown*, U_k, in our case). Other similarity metrics can be used, but Equation (5.5) was enough to achieve good results in our experimental evaluation.

The execution of Algorithm 9 is illustrated in the following example.

Example 1. *Suppose that the source and target tasks have the classes* $C^{source} = \{Class1, Class2\}, C^{target} = \{Class1, Class2, Class3\}$, *with* $\mathscr{X}_C(Class1) = Class1$, *and* $\mathscr{X}_C(Class2) = Class2$. *The attributes are* $Att^{source}(Class1) = \{a1, a2\}$, $Att^{target}(Class1) = \{a1\}$, $Att^{source}(Class2) = \{b1\}$, $Att^{target}(Class2) = \{b1, b2\}$, *and* $Att(Class3) = \{c1\}$, *with domains* $Dom^{source}(a1) = Dom(a2) = \{0, 1, U_k\}$, $Dom^{target}(a1) = \{0, 2\}$, $Dom^{source}(b1) = \{0, 1\}$, $Dom^{target}(b1) = Dom(b2) = \{0, 1\}$, *and* $Dom(c1) = \{0, 1\}$. $O^{source} = \{o1_{s1}, o1_{s2}, o2_{s2}\}$, $O^{target} = \{o1_{t1}, o2_{t1}, o1_{t2}, o2_{t2}, o3_{t2}, o1_{t3}\}$, *where the index* $s1$ *means that the object belongs to* $Class1$ *from the source task and* $t1$ *means that the object belongs to* $Class1$ *from the target task. In the target task we want to translate the state* $s_t = \{\langle 0\rangle, \langle 2\rangle, \langle 0, 1\rangle, \langle 1, 0\rangle, \langle 0, 0\rangle, \langle 0\rangle\}$. *When building* P_{Class1}, $n_t = 2$ *and* $n_s = 1$, *and hence the combinations are* $P_{Class1} = \{(\langle 0\rangle), (\langle 2\rangle)\}$. *On its turn,* $n_t = 3$ *and* $n_s = 2$ *when building* $P_{Class2} = \{(\langle 0, 1\rangle, \langle 1, 0\rangle), (\langle 0, 1\rangle, \langle 0, 0\rangle), (\langle 1, 0\rangle, \langle 0, 0\rangle)\}$. $Class3$ *is ignored because it does not exist in the source task. The assemble operation then builds states by combining each possible elements of* P_{Class1} *and* P_{Class2}, *resulting in* $\Theta = \{(\langle 0\rangle, \langle 0, 1\rangle, \langle 0, 0\rangle), (\langle 0\rangle, \langle 0, 1\rangle, \langle 0, 0\rangle), \ldots, (\langle 2\rangle, \langle 1, 0\rangle, \langle 0, 0\rangle)\}$. *However, when computing the similarity metric,* $o2_{t1}$ *has value* $o2_{t1}.a1 = 2 \notin Dom^{source}(a_1)$. *Therefore, all states containing that object have* 0 *similarity value and are removed from the mapping. The similarity function ignores* $b2$ *(because it does not exist in the source task) and initiates* $a2$ *with a standard value (because it does not exists in the target task). The following states have similarity value equals to* 1*:* $(\langle 0, U_k\rangle, \langle 0\rangle, \langle 1\rangle)$, $(\langle 0, U_k\rangle, \langle 0\rangle, \langle 0\rangle)$, *and* $(\langle 0, U_k\rangle, \langle 1\rangle, \langle 0\rangle)$. *Thus, these states are mapped with probability* $\frac{1}{3}$.

In the next section we describe approaches to transfer Q-values using a PITAM.

5.2.1 Transferring Knowledge using PITAM

After defining $\mathscr{P}_o$, the agent can make use of the mapping to transfer knowledge from one task to another. Here, we assume that the learning agent uses a TD algorithm (thus a Q-function is available) and that a mapping $\mathscr{X}_a$ for actions is given (Even though learning $\mathscr{X}_a$ is possible (TAYLOR; WHITESON; STONE, 2007), so far no

Algorithm 9 Zero-Shot Autonomous Mapping Learning

Require: Set of objects in both tasks O^{source} and O^{target}, set of classes C^{source}, class mapping $\mathscr{X}_C$, current object-oriented state s_t in the target task.

for $\forall C_i \in C^{source}$ **do**
  $C_t \leftarrow \mathscr{X}_C(C_i)$
  // number of objects belonging to C_i in each task.
  $n_t \leftarrow numObj(O^{target}, C_t)$
  $n_s \leftarrow numObj(O^{source}, C_i)$
  // n_s-combinations of objects from the target task.
  $P_{C_i} \leftarrow combinations(s_t, n_t, n_s, C_t)$
// Assembles all possible states using objects from P.
$\Theta \leftarrow assemble(P)$
// Sum of similarity values.
$totalSim \leftarrow 0$
for $\forall s_s \in \Theta$ **do**
  // Equation (5.5)
  $totalSim \leftarrow totalSim + similarity(s_t, s_s)$
for $\forall s_s \in \Theta$ **do**
  $\mathscr{P}_o(s_t, s_s) \leftarrow \frac{similarity(s_t, s_s)}{totalSim}$

algorithm can learn such a mapping without samples from the target task).

We here propose two strategies to initiate the Q-function in the target task: *QAverage* and *QBias*. In both cases, the initialization is intended to better guide the exploration during learning. Algorithm 10 describes how this initialization works for both strategies. For each possible state in the target task, the agent defines the object-oriented representation of this state, and then uses it to get a mapping ω to object sets in the source task. ω is a set of tuples $\langle s_s, p \rangle$, where s_s is the object-oriented state in the source task and p is the probability defined by PITAM $\mathscr{P}_O$. ω is then used to define the initial value of the Q-function to s_t, which here is calculated for each strategy as follows:

1. **QAverage**: Here, all states with a non-zero probability defined by $\mathscr{P}_O$ contribute to a resulting Q-value, proportionally to their PITAM probability.

$$initQ(\omega, a) = \sum_{\langle s_s, p \rangle \in \omega} Q^{source}(s_s, \mathscr{X}_{\mathscr{A}}(a))p. \tag{5.6}$$

2. **QBias**: Alternatively, we can initiate only the Q-value of the best action with a small bias value b (BOUTSIOUKIS; PARTALAS; VLAHAVAS, 2011). The main idea here is to initiate in the new domain reusing the optimal policy, but at the same time introducing only a small value in the Q-table to enable faster

Algorithm 10 Q-table initialization

Require: Set of possible states S^{target} and S^{source}, action mapping $\mathscr{X}_A$, PITAM $\mathscr{P}_O$.

```
for ∀s_t ∈ S^target do
    ω ← ∅
    // For all mapped states with a non-zero probability.
    for ∀s_s ∈ S^source, P_O(s_t, s_s) > 0 do
        // Stores mappings to states in the source task.
        ω ← ω ∪ ⟨s_s, P_O(s_t, s_s)⟩
    for ∀a ∈ A^target do
        // Calculates the resulting Q-value.
        Q(s_t, a) ← initQ(ω, X_A(a))                    ▷ Eq (5.6) or (5.7).
```

policy refinements.

$$initQ(\omega, a) = \begin{cases} b & \text{if } \mathscr{X}_A(a) = \\ & \arg\max_{a_s \in A^{source}} \sum_{\langle s_s, p \rangle \in \omega} Q^{source}(s_s, a_s). \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

The standard value for uninitialized Q-values is 0. Through the combination of PITAM with one of the aforementioned initiation strategies, we can autonomously transfer knowledge across tasks with no need of gathering samples of interactions in the target task. In the next sections we describe the Experimental evaluation to show the benefits of our proposal.

5.2.2 Experimental Evaluation

In order to evaluate the effectiveness of our proposal under several conditions, we test the speed-up provided by the autonomously learned PITAM in varied modifications of two benchmark domains: *Predator-Prey* and *Goldmine*. The following algorithms were compared in our experiments: (i) **Regular Learning**: The standard Q-Learning algorithm; (ii) **QAverage**: An autonomously learned PITAM with the *QAverage* knowledge reuse strategy following Equation (5.6); (iii) **QBias**: Another PITAM-based algorithm, following the Q-table initialization described by Equation (5.7); and (iv) **QManualMapping**: A manually defined Inter-Task Mapping using Q-Value Reuse.

For all experiments we set $\alpha = 0.1$, $\gamma = 0.9$, and all algorithms use a ϵ-greedy exploration strategy with an initial $\epsilon = 0.5$ with a decay of 0.999 applied after each episode. The majority of our experiments was executed in the *Predator-Prey* domain (TAN, 1993; BOUTSIOUKIS; PARTALAS; VLAHAVAS, 2011). In order to test the

Table 5.1. Manual Inter-Task Mapping translating the target task with 2 preys and 4 predators to a source task with 1 prey and 3 predators. Here, z is the local agent, *Prey* is the set of prey objects, *Predator* is the set of predator objects, and *dist* is an Euclidian distance function.

source task	**target task**
prey1	$\underset{p \in Prey}{arg\,\min}\ dist(z, p)$
predator1	$p1 = \underset{p \in Predator}{arg\,\min}\ dist(z, p)$
predator2	$\underset{p \in (Predator - p1)}{arg\,\min}\ dist(z, p)$

effectiveness of our proposal when transferring knowledge across tasks, we also consider the *Goldmine* domain in our experimental evaluation. Notice that the *Goldmine* and *Predator-Prey* tasks are similar, but *Goldmine* is much easier to solve because the environment is stationary and gold pieces do not move. Thus, the optimal policy is to move towards gold pieces as soon as they are seen.

For all experiments, 100 evaluations episodes were carried out without exploration for each 10 learning episodes. For all source tasks, 2500 learning episodes were executed and then the learned Q-table is stored for posterior reuse. For all target tasks, 4500 learning episodes are executed in total. In all cases, when the agent is in a blind state (no observation is received), a random action is executed. All statistical significance tests were carried out by a 95% confidence Wilcoxon signed-rank test.

Experiment 1 - Scaling Tasks: In this first experiment we evaluate the effectiveness of each TL approach when the source and target tasks are in the *Predator-Prey* domain and differ only in attribute domains and number of objects, that is, the visual depth and number of predators/preys are increased without any change in transition and reward functions. For both source and target tasks, all predators receive +1 of reward when a prey is captured, and 0 otherwise. While in the source task we train 3 predators trying to catch 1 prey with $depth = 3$, the target task has 4 predators trying to catch 2 preys with $depth = 4$. As we have no changes in the class and action sets, the mapping $\mathcal{X}_C$ for PITAM is trivially a direct translation, and the mapping for *QManualMapping* is defined according to Table 5.1. Notice that this mapping relies on a very domain-specific knowledge that the closest prey is more important to the value function, and such domain knowledge is not needed for PITAM.

Experiment 2 - Changes in Reward Functions: Here we evaluate the algorithms when the reward function of the target task is different from the source task (without many differences in the optimal policy). While the source task is the same as Experiment1, in the target task agents receive +100 of reward when a prey is captured, and $\frac{\sqrt{100}}{dist}$ otherwise, where *dist* is the distance to the closest prey.

Experiment 3 - Changes in Transition Functions: Here we change the state transition function across tasks. While the source task and reward functions are the same as in Experiment1, the action effects are depicted in Figure 5.8.

Experiment 4 - TL Across Tasks: Here we take as source task the *Goldmine* task described in Section 2.7.3. The target task is defined as in Experiment1. The Class Mapping is defined as $\mathcal{X}_O(Prey) = Gold$, $\mathcal{X}_O(Predator) = Miner$. *QManualMapping* maps the closest predators to the closest miners, and the closest preys to the closest gold pieces, similarly as in Table 5.1.

5.2.3 Results

First we train a learning agent in the *Predator-Prey* task to be used as source for experiments 1-3. Figure 5.9a depicts the average number of steps to solve the task during learning. Even though the agent has not learned the optimal policy after episode 2500, the observed performance shows that it is possible to learn an effective policy in this task, as the performance achieved by the agent after the learning process is much better than of a random agent. For the sake of simplicity we refer to the performance after 4500 learning episodes as asymptotic performance. Although convergence is not achieved yet after this amount of learning episodes, only very small changes in performance are observed after it.

Experiment 1: In this experiment we evaluate the performance of TL when the target task is a scaled version of the source task. Figure 5.10 shows the performance of each TL algorithm compared with regular learning. All TL algorithms present advantages in terms of jumpstart. While *Regular learning* (at first equivalent to a random actuation) solves the task after 76.21 steps on average, *QManualMapping*, *QAverage*, and *QBias* solve the task, respectively, in 62.43, 40.71, and 45.48 steps on average. Even though all TL techniques have a significantly better performance right after the

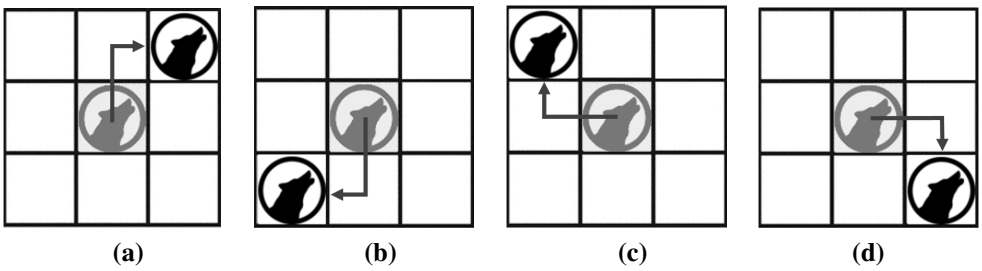


Figure 5.8. Predator movement after using actions *North* (a), *South* (b), *West* (c), and *East* (d) in Experiment 3.

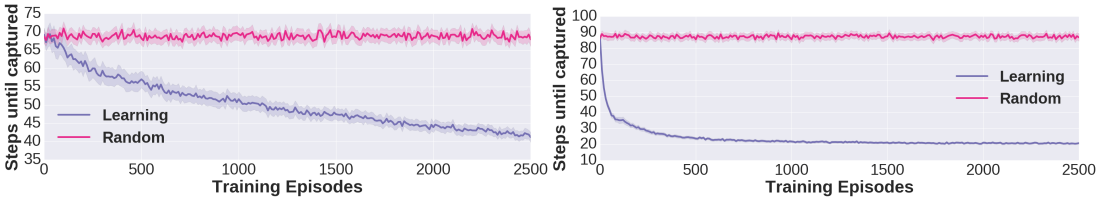


Figure 5.9. The average of steps to solve the task during evaluation episodes in 100 repetitions for: (a) *Predator-Prey* and (b) *Goldmine* domains. The shaded area corresponds to the 99% confidence interval. The performance of a random agent is included as baseline.

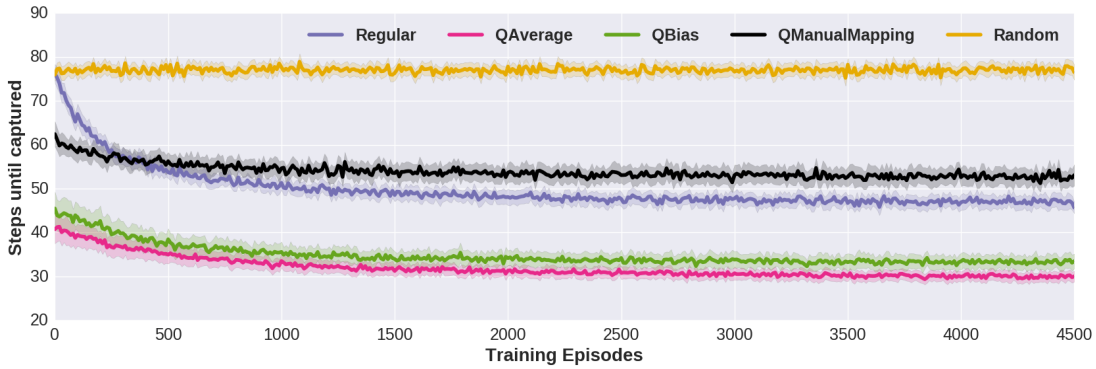


Figure 5.10. The average performance 50 repetitions of Experiment 1. The shaded area corresponds to the 99% confidence interval. The performance of a random agent is included as baseline.

beginning in the new task, *QManualMapping* takes longer to improve its policy in the new task because the original Q-values in the source task are still used and harder to be replaced. The difference between *Regular learning* and *QManualMapping* is not statistically significant between 240 and 450 learning episodes, and *Regular learning* achieves a significantly better performance for the rest of learning process. On their turn, *QBias* and *QAverage* have a better asymptotic performance, roughly stabilizing on 33.50 and 30.03 steps, against the *Regular learning* performance of 45.67 steps. The difference between *QBias* and *QAverage* is statistically significant through the entire experiment. Note that, after 1500 learning episodes, the performance for *Regular learning* roughly plateaus before achieving the performance of PITAM-based algorithms.

Experiment 2: In this experiment the reward function was modified while the optimal policy remains similar across tasks. Figure 5.11 shows that the relative results are very similar to the first experiment, which shows that all the evaluated TL approaches are robust in terms of simple changes of scale in reward functions across tasks. The main differences between experiments 1 and 2 are that here the standard deviation is higher, and the reward function in this task is less effective to guide the agent towards

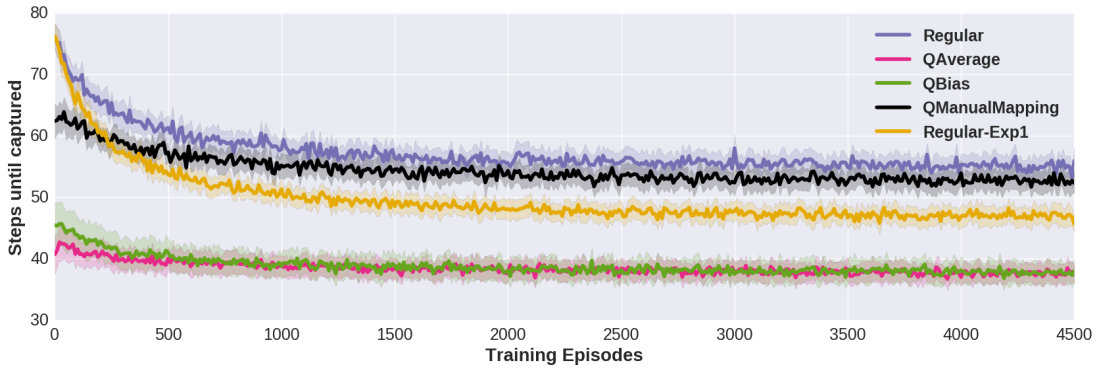


Figure 5.11. The average performance 50 repetitions of Experiment 2. The shaded area corresponds to the 99% confidence interval. The performance of regular learning in Experiment 1 was included to allow comparisons in regard to reward function effectiveness.

the task solution (which is evidenced when comparing the *Regular learning* performance in both experiments). However, the TL algorithms remain better than *Regular learning*. *QAverage* and *QBias* are again better in both jumpstart and asymptotic performance. The difference between *QAverage* and *QBias* is only significant until episode 200. After that, they have a equivalent performance. Here, *QManualMapping* is significantly better than *Regular learning* during the whole experiment.

Experiment 3: The third experiment considers changes in the transition function, which are here implemented as changes in action effects. Although different, the source and target tasks are still related because the agent moves towards the intended direction, with an additional collateral effect of moving in some direction in another axis. The results shown in Figure 5.12 depict that, in this case, *QManualMapping* presented no advantages over *Regular learning*. Until learning step 60, the performance of *QManualMapping* was not significantly different from *Regular learning*, and after that *QManualMapping* was significantly worse than all other algorithms. *QBias* is significantly better than *Regular learning* in jumpstart, but *QAverage* is not, and they are consistently better after 1000 learning episodes, which shows that the TL initialization provided a better exploration than learning from scratch. *QBias* and *QAverage* have similar performances during the learning process and are not significantly different in asymptotic performance.

Experiment 4: The last experiment accomplishes TL through different yet similar domains. Here, the source task is learned in the *Goldmine* domain. Comparing Figures 5.9a and 5.9b, we can see that the *Goldmine* task is simpler to solve and thus being able to reuse its solution in another (more complex) task is very interesting. Figure 5.13 shows that using a simpler version of the target task presents very good results for TL. Again, all TL algorithms presented benefits in regard to jumpstart, showing that simply

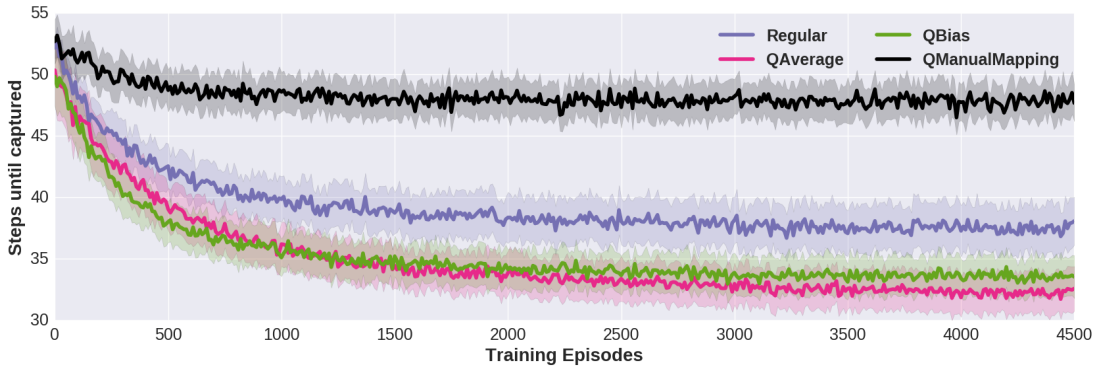


Figure 5.12. The average performance 75 repetitions of Experiment 3. The shaded area corresponds to the 99% confidence interval.

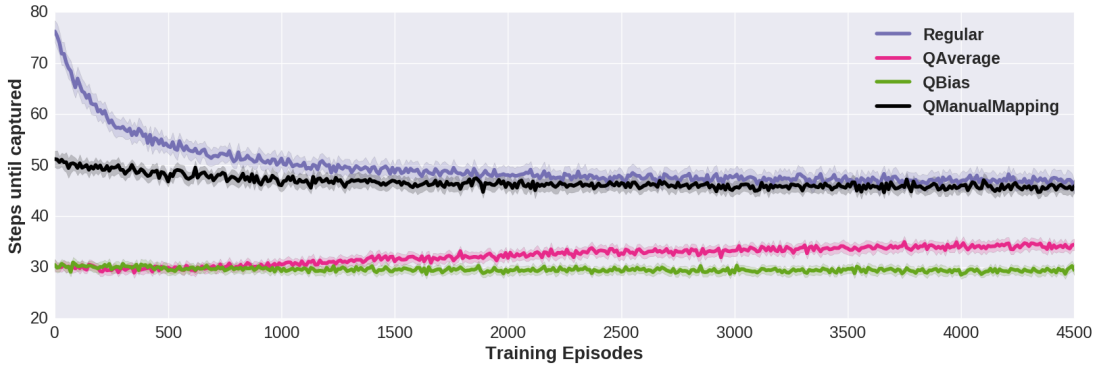


Figure 5.13. The average performance 50 repetitions of Experiment 4. The shaded area corresponds to the 99% confidence interval.

following a prey is a very effective strategy. *QBias* and *QAverage* present better results in asymptotic performance, showing that the initialization provided by TL greatly improved the learned policy over time. *QAverage* losses some performance when trying to adapt in the new task, however the final performance is still far better than *Regular learning* and *QManualMapping*. *QManualMapping* and *Regular learning* have equivalent performances after 2000 learning steps. *QBias* has the best performance in this experiment by maintaining the performance achieved when transferring the Q-table from the source task, as the difference between *QBias* and *QAverage* is statistically significant from 1300 learning episodes until the end of the training.

Table 5.2 summarizes the results of each algorithm in all experiments. Notice that PITAM approaches (QBias and QAverage) were never worse than *Regular learning* and Q-value reuse with a hand-coded Inter-Task Mapping. The results discussed in this paper show that PITAM is a promising method, and can be a determinant step towards bulding agents capable of autonomously reusing knowledge across tasks.

Table 5.2. Summary of all experimental results. The value here represented are the average number of steps to solve the task in each experiment, where *jumpstart* is the performance after 0 learning episodes and *asymptotic* is the performance after 4500 episodes.Best results (according to statistical significance tasks) are denoted in bold.

		Regular	QManualMapping	QAverage	QBias
Experiment 1	jumpstart	76.21	62.43	**40.75**	45.48
	asymptotic	45.67	53.09	**30.03**	33.50
Experiment 2	jumpstart	76.21	62.43	**40.75**	45.48
	asymptotic	55.97	52.34	**37.48**	**38.04**
Experiment 3	jumpstart	52.18	52.70	50.35	**49.83**
	asymptotic	38.08	47.68	**32.59**	**33.54**
Experiment 4	jumpstart	76.21	51.22	**30.34**	**30.48**
	asymptotic	45.67	45.92	34.48	**29.48**

5.2.4 Discussion

Being able to find correspondences between tasks is a key issue to successfully transfer knowledge across tasks. We here rely on object-oriented task descriptions to autonomously define Probabilistic Inter-TAsk Mappings (PITAM), which estimates how similar two given states in the target and source tasks are. We also proposed two methods to reuse Q-table estimates across tasks using the learned PITAM. Our preliminary experiments included evaluations when transferring knowledge between tasks with different state spaces, reward functions, transition functions, and different domains. When compared to regular learning and *Q-value Reuse* a hand-coded Inter-Task Mapping , our proposal presented benefits in all evaluated scenarios, even though less domain knowledge is needed.

5.3 Curriculum Generation

As discussed in Section 2.5, *Curriculum* Learning might enable a learning agent to master a complex task faster by solving portions of it individually and reusing the knowledge (as long as proper task generation, knowledge reuse, and task ordering strategies are available). In this section we show how the object-oriented task description can be leveraged to perform the task generation and task ordering steps in a *Curriculum* Learning approach. This work was previously published in a conference paper (SILVA; COSTA, 2018).

5.3.1 Object-Oriented Curriculum Generation

As shown in the previous sections, object-oriented descriptions can be used to provide generalization opportunities. We here contribute a method to take advantage of this generalization to facilitate *Curriculum* construction and use.

Since all tasks within a *Curriculum* are in the same domain, we have a single set of classes for all tasks, but each task may have a different set of objects and initial state. Thus, for our purposes we define an *Object-Oriented* task $\mathcal{T}_i$ as:

$$\mathcal{T}_i = \langle \boldsymbol{C}, \boldsymbol{O}^{\mathcal{T}_i}, S_0^{\mathcal{T}_i}, A^{\mathcal{T}_i}, T^{\mathcal{T}_i}, R^{\mathcal{T}_i} \rangle \tag{5.8}$$

where $\boldsymbol{C}$ is the set of classes, $\boldsymbol{O}^{\mathcal{T}_i}$ is the set of objects in $\mathcal{T}_i$, $S_0^{\mathcal{T}_i} : S \rightarrow [0, 1]$ is the probability of task $\mathcal{T}_i$ starting in each state, $A^{\mathcal{T}_i}$ is the set of possible actions for $\mathcal{T}_i$, $T^{\mathcal{T}_i}$ is the transition function, and $R^{\mathcal{T}_i}$ is the reward function. If attributes not related to objects are required for $\mathcal{T}_i$, an *environment* class can be created for adding those attributes, hence the definition of *degrees of freedom* is no longer necessary. As in (SVETLIK et al., 2017), we consider a *Curriculum* as a graph of tasks $\mathbb{C} = \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$. For building $\mathbb{C}$, we use a strategy similar to Svetlik's (SVETLIK et al., 2017), adding in the *Curriculum* tasks with high transfer potential, which we here calculate based on the *Object-Oriented* description as:

$$\nu(\mathcal{T}_s, \mathcal{T}_t, \boldsymbol{\mathcal{T}}_{\mathbb{C}}) = \frac{sim_{\mathbb{C}}(\mathcal{T}_s, \mathcal{T}_t)}{\max_{\mathcal{T}_i \in \mathcal{T}_{\mathbb{C}}} sim_{\mathbb{C}}(\mathcal{T}_s, \mathcal{T}_i)} . \frac{|S_t|(|\boldsymbol{O}^{\mathcal{T}_t}| + 1)}{|S_s|(|\boldsymbol{O}^{\mathcal{T}_s}| + 1)} \tag{5.9}$$

where $\mathcal{T}_s$ is the task to measure the transfer potential, $\mathcal{T}_t$ is the target task, $\boldsymbol{\mathcal{T}}_{\mathbb{C}}$ is the set of tasks already defined to be executed[1] before $\mathcal{T}_t$, $sim_{\mathbb{C}}(\mathcal{T}_i, \mathcal{T}_j)$ is a similarity value between tasks $\mathcal{T}_i$ and $\mathcal{T}_j$, $|S_i|$ is the size of the state space in $\mathcal{T}_i$, and $\boldsymbol{O}^{\mathcal{T}_i}$ is the set of objects of task $\mathcal{T}_i$.

The intuition behind this equation is to prioritize the inclusion of tasks that are: (i) similar to the target task, increasing the usefulness of the learned policy; (ii) dissimilar from the previously included tasks, to reduce the amount of redundant knowledge; and that (iii) have a smaller state space than in the target task, to first train in smaller tasks. Here, the similarity between tasks is calculated as:

$$sim_{\mathbb{C}}(\mathcal{T}_s, \mathcal{T}_t) = \sum_{C_i \in \boldsymbol{C}} \frac{|\boldsymbol{O}_{C_i}^{\mathcal{T}_s} \cap \boldsymbol{O}_{C_i}^{\mathcal{T}_t}|}{\max(|\boldsymbol{O}_{C_i}^{\mathcal{T}_s}|, |\boldsymbol{O}_{C_i}^{\mathcal{T}_t}|)} + \frac{|S_s \cap S_t|}{|S_t|} \tag{5.10}$$

where $\mathcal{T}_s$ and $\mathcal{T}_t$ are tasks from the same domain, C_i is one class from the set $\boldsymbol{C}$,

[1] If $\boldsymbol{\mathcal{T}}_{\mathbb{C}} = \emptyset$, consider $\max_{\mathcal{T}_i \in \mathcal{T}_{\mathbb{C}}} sim_{\mathbb{C}}(\mathcal{T}_s, \mathcal{T}_i) = 1$.

$\boldsymbol{O}_{C_i}^{\mathcal{T}_s} \cap \boldsymbol{O}_{C_i}^{\mathcal{T}_t}$ is the set of objects that belong to class C_i and have the same attribute values in the initial state for both $\mathcal{T}_s$ and $\mathcal{T}_t$, and $|S_s \cap S_t|$ is an estimate of the number of states the two tasks have in common[2]. Notice that the real similarity between two tasks cannot be computed because it depends on the transition and reward functions that are unknown to the agent. The proposed equation gives a good indication of the true similarity if: (i) object attributes have the same semantic meaning in both tasks; and (ii) similar tasks in this domain have some objects in common. We believe that those are reasonable assumptions for the purpose of building *Curricula*.

We generate $\mathbb{C}$ by following Algorithm 11. The set of source tasks $\boldsymbol{\mathcal{T}}$ to be used as a parameter has been manually defined by humans in the works so far, but an automatically generated set could also be used (as in our proposed method detailed in the next section).

Algorithm 11 Automatic Curriculum Generation

Require: set of source tasks $\boldsymbol{\mathcal{T}}$, target task $\mathcal{T}_t$, threshold for task inclusion ϵ.

1: $\boldsymbol{\mathcal{V}} \leftarrow \emptyset, \boldsymbol{\mathcal{E}} \leftarrow \emptyset$ ▷ Initializing the Curriculum Graph
2: $(\boldsymbol{G}, \boldsymbol{\mathcal{T}}_{\mathbb{C}}) = groupTasks(\boldsymbol{\mathcal{T}}, \mathcal{T}_t, \epsilon)$ ▷ Algorithm 12
3: $\boldsymbol{\mathcal{V}} \leftarrow \boldsymbol{\mathcal{T}}_{\mathbb{C}} \cup \mathcal{T}_t$
4: **for** $\forall \boldsymbol{g} \in \boldsymbol{G}$ **do** ▷ Intra-group Transfer
5: **for** $\forall \mathcal{T}_i \in \boldsymbol{g}$ **do**
6: $\mathcal{T}_m = \arg\max_{T_j \in \boldsymbol{g} | j>i} \nu(\mathcal{T}_j, \mathcal{T}_i, children(\mathcal{T}_i))$ ▷ Eq. (5.9)
7: **if** $\nu(\mathcal{T}_m, \mathcal{T}_i, children(\mathcal{T}_i)) > \epsilon$ **then**
8: $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{E}} \cup \langle \mathcal{T}_m, \mathcal{T}_i \rangle$
9: **for** $\forall \boldsymbol{g} \in \boldsymbol{G}$ **do** ▷ Inter-group Transfer
10: **for** $\forall \boldsymbol{g}' : \boldsymbol{g}' \in \boldsymbol{G}$ and $\boldsymbol{g}.features \subset \boldsymbol{g}'.features$ **do**
11: **for** $\forall \mathcal{T}_i \in \boldsymbol{g}$ **do**
12: $\mathcal{T}_m = \arg\max_{T_j \in \boldsymbol{g}'} \nu(\mathcal{T}_j, \mathcal{T}_i, children(\mathcal{T}_i))$ ▷ Eq. (5.9)
13: **if** $\nu(\mathcal{T}_m, \mathcal{T}_i, children(\mathcal{T}_i)) > \epsilon$ **then**
14: $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{E}} \cup \langle \mathcal{T}_m, \mathcal{T}_i \rangle$
15: **for** $\forall \mathcal{T}_i : \mathcal{T}_i \in \boldsymbol{\mathcal{T}}_{\mathbb{C}}$ and $deg^+(\mathcal{T}_i) = 0$ **do** ▷ Add edges to target
16: $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{E}} \cup \langle \mathcal{T}_i, \mathcal{T}_t \rangle$
17: $\mathbb{C} \leftarrow \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$
18: return $\mathbb{C}$

The first step is to split the set of source tasks $\boldsymbol{\mathcal{T}}$ in groups according to their parameters (line 2). The grouping procedure has two purposes: (i) discarding tasks that are not expected to be beneficial for transfer and (ii) organizing $\boldsymbol{\mathcal{T}}$ into groups of tasks with the same parameters. Imagine that $\mathcal{T}_t$ is a robot soccer task against two opponents. A possible resulting set of groups would be: one group composed of tasks with no opponents; one composed of one-opponent tasks; and one with two-opponent

[2]This estimate is computed by evaluating the intersection of the object attribute *domains* in the tasks. Thus, no additional knowledge is needed.

tasks, where the tasks in which the initial states are very different from the one in $\mathcal{T}_t$ are discarded. Our grouping procedure is fully described in Algorithm 12 and the main idea is to build a set of candidate tasks $\boldsymbol{\mathcal{T}}_{\mathbb{C}}$ discarding tasks with low *Object-Oriented* transfer potential (Equation (5.9)).

Algorithm 12 groupTasks

Require: set of source tasks $\boldsymbol{\mathcal{T}}$, target task $\mathcal{T}_t$, threshold for task inclusion ϵ.

1: $\boldsymbol{\mathcal{T}}_{\mathbb{C}} \leftarrow \emptyset$
2: $\boldsymbol{G} \leftarrow \emptyset$
3: **for** $\forall \mathcal{T}_i \in \boldsymbol{\mathcal{T}}$ **do**
4: **if** $\nu(\mathcal{T}_i, \mathcal{T}_t, \boldsymbol{\mathcal{T}}_{\mathbb{C}}) > \epsilon$ **then** ▷ Eq. (5.9)
5: **if** $\nexists \boldsymbol{g} : \boldsymbol{g} \in \boldsymbol{G}$ and $\boldsymbol{g}.features = \mathcal{T}_i.features$ **then**
6: $\boldsymbol{G} \leftarrow \boldsymbol{G} \cup group(\mathcal{T}_i.features)$ ▷ New group
7: $\boldsymbol{g} = \{\boldsymbol{g} \in \boldsymbol{G} | \boldsymbol{g}.features = \mathcal{T}_i.features\}$
8: $\boldsymbol{g} \leftarrow \boldsymbol{g} \cup \mathcal{T}_i$
9: $\boldsymbol{\mathcal{T}}_{\mathbb{C}} \leftarrow \boldsymbol{\mathcal{T}}_{\mathbb{C}} \cup \mathcal{T}_i$
10: Sort tasks within each group ($\mathcal{T}_i \in \boldsymbol{g}$) by $\nu(\mathcal{T}_i, \mathcal{T}_t)$
11: return $(\boldsymbol{G}, \boldsymbol{\mathcal{T}}_{\mathbb{C}})$

Algorithm 12 returns a group for each possible task parameter in which there exists at least one task with transfer potential higher than threshold ϵ (Alg 12 line 4).

After the set of task groups $\boldsymbol{G}$ is built, we define the edges of the *Curriculum* as proposed by Svetlik *et al.* (SVETLIK et al., 2017). Firstly we search for pairs of tasks that have a high transfer potential between themselves within the same task group (Alg. 11 lines 4-8), then we search for pairs that belong to different groups, as long as the features of one task are contained in another's (lines 9-14). Here, operation $children(\mathcal{T}_i)$ returns the set of tasks already included in the *Curriculum* that have an edge to $\mathcal{T}_i$. In the example of a Robot Soccer task, $\boldsymbol{g}.features \subset \boldsymbol{g'}.features$ if the number of opponents in tasks inside $\boldsymbol{g}$ is smaller than for the tasks inside $\boldsymbol{g'}$. Hence, the knowledge from the simpler tasks will be transferred to the most complex ones. Finally, we create an edge from the tasks with outdegree *zero*[3] to the target task (line 16).

As an alternative for executing all tasks in the *Curriculum*, which is ineffective for big *Curricula*, we propose a novel procedure to use only portions of a *Curriculum* (Algorithm 13). We use a biased *Random Walk* to traverse the *Curriculum* graph $\mathbb{C}$ starting from the target task (a back-to-front path - line 5), without, however, visiting every vertex in $\mathbb{C}$. Then, we walk n_{steps} steps from $\mathcal{T}_t$, biasing each step according to the transfer potential of each of its *children*, i.e., tasks with high transfer potential are

[3] outdegree *zero* means that no edge is starting from the task, i.e., it still has no parents.

more likely to be selected (lines 5-9):

$$p(\mathcal{T}_i) = \frac{\nu(\mathcal{T}_i, curTask, \boldsymbol{T})}{\sum_{\mathcal{T}_j \in children(curTask)} \nu(\mathcal{T}_j, curTask, \boldsymbol{T})}. \tag{5.11}$$

Each of the selected tasks are labeled[4] for posterior use, and we keep a list of selected tasks for computing the transfer potential (lines 7 and 9). This procedure is repeated n_{traj} times. Finally, all labeled tasks are used for learning, finishing in the target task (lines 10-13).

Algorithm 13 *learn* with biased *Random Walk*.

Require: *Curriculum* $\mathbb{C} = \{\boldsymbol{\mathcal{V}}, \boldsymbol{\mathcal{E}}\}$, target task $\mathcal{T}_t$, n_{traj}, n_{steps}.

1: $\boldsymbol{T} \leftarrow \emptyset$
2: **for** n_{traj} times **do**
3: $curTask \leftarrow \mathcal{T}_t$
4: $label(curTask)$
5: **for** n_{steps} times **do**
6: Draw $\mathcal{T}_i \in children(curTask)$ according to Eq. (5.11)
7: $label(\mathcal{T}_i)$
8: $curTask \leftarrow \mathcal{T}_i$
9: $\boldsymbol{T} \leftarrow \boldsymbol{T} \cup \mathcal{T}_i$
10: **while** $\exists labeled(\mathcal{T}_i), \forall \mathcal{T}_i \in \boldsymbol{T}$ **do**
11: Select a labeled task $\mathcal{T}_i$ with no labeled children
12: Learn $\mathcal{T}_i$ reusing previous knowledge
13: $unlabel(\mathcal{T}_i)$

Then, a procedure for reusing knowledge from the previously solved tasks must be defined. We use a method based on value function reuse as performed by Narvekar *et al.* (NARVEKAR et al., 2016), but taking advantage of the *Object-Oriented* representation. For that, we firstly map the current state to a set of similar states in the source tasks. Then, we reuse their value functions in the new Q-table. This mapping is calculated with PITAM $\mathscr{P}$, as proposed in the last section. For defining $\mathscr{P}$, a similarity metric sim_{PITAM} is calculated between the current state and all the state space in the source task by using the *Object-Oriented* description. Then, $\mathscr{P}$ is calculated by normalizing the similarity values into probabilities: $\sum_{s \in S_{\mathcal{T}_s}} \mathscr{P}_{\mathcal{T}_s, \mathcal{T}_t}(s_t, s) = 1$. We here define the similarity value as : $sim_{PITAM}(s_t, s_s) = |\boldsymbol{O}_{s_t} \cap \boldsymbol{O}_{s_s}|$, where $|\boldsymbol{O}_{s_t} \cap \boldsymbol{O}_{s_s}|$ denotes the number of objects that belong to the same class and have the same attribute values for two particular states s_t and s_s. That is, states that have no object in common have a mapping with zero probability, and the higher the number of common objects, the higher the mapping probability will be. Although calculating exact PITAM probabilities is computationally expensive, it is easy to use domain knowledge to prune the

[4] A *label* here is binary mark on a vertex, posteriorly used for defining the tasks that were chosen for execution.

state space search (e.g., calculating similarity values only for states in which there are objects with similar attribute values). After the PITAM calculation, we initialize the Q-table in the new task $\mathcal{T}_t$ as:

$$Q_{\mathcal{T}_t}(s_t, a_t) \leftarrow \frac{\sum_{\mathcal{T}_s \in \mathbb{C}_{\mathcal{T}_t}} \sum_{s_s \in S_{\mathcal{T}_s}} \mathscr{P}_{\mathcal{T}_s,\mathcal{T}_t}(s_t, s_s) Q_{\mathcal{T}_s}(s_s, a_t)}{|\mathbb{C}_{\mathcal{T}_t}|}, \tag{5.12}$$

where $\mathbb{C}_{\mathcal{T}_t}$ is the set of already solved tasks that had an edge to $\mathcal{T}_t$ in the original *Curriculum* graph. Notice that the object attributes must be agent-centered to facilitate transfer. For example, if the agent is in a world represented by a grid, it is very hard to generalize states if the object observations are given by their absolute positions. However, if distances between the agent and objects are used as attributes (agent-centered representation), it is much easier to find state correspondences (i.e., find objects with the same attribute values in two states).

Using the *Curriculum* and this transfer procedure, the agent is expected to learn faster than when learning from scratch. In the next Section we describe our proposal to automatically generate source tasks ($\boldsymbol{\mathcal{T}}$ in Algorithm 11).

5.3.2 Automatic Source Task Generation

The success of a *Curriculum* depends on: (i) a proper set of source tasks; (ii) a proper task ordering; (iii) the efficacy of the chosen TL algorithm; (iv) a good stopping criterion to identify when to switch tasks. Even though Svetlik *et al.* (SVETLIK et al., 2017) had proposed a method to define the sequence, automatically defining the set of source tasks was still an open problem (SVETLIK et al., 2017; NARVEKAR et al., 2016; PENG et al., 2016b).

We here propose a method to generate source tasks by using the *Object-Oriented* representation, requiring less human effort than previous works in which this set must be manually given. The intuition of our proposal is to randomly select a portion of the objects from the target task to build each of the (smaller) source tasks, assuming that smaller tasks are easier to solve.

Algorithm 14 fully describes our source task generation procedure. At first we define the set $\boldsymbol{F}_{simple}$, which is a set of possible valuations for the *environment* objects in the target task (line 2). We assume that all domains have an *environment* class C_{Env}, which has as attributes domain features not related to other classes (e.g., the grid size in a *Gridworld*). The $simplify$ function creates environment objects corresponding to simplified versions of the target task, for example, if the environment object is the size of a grid, a possible simplification would be a set of values corresponding to

smaller grids. Then, we define the set of objects belonging to the class with fewer objects among the remaining classes C_{min} (lines 3–4). We then create o_{min} tasks, each of them containing from 0 to o_{min} objects from C_{min} (line 6), one possible set of values from $\boldsymbol{F}_{simple}$ (line 7), and a random number of objects from the other classes (line 11). The initial state for this new source task if defined through the *initState* function. A possible way to implement this function is to draw random attribute values for all objects, or copy the values from the target task initial state (when applicable). The action space, transition function, and reward functions for the new task are then defined through the *actionSpace*, *transitionFunction*, and *rewardFunction* functions[5], and the task is finally added to the set of source tasks $\boldsymbol{\mathcal{T}}$. This process is repeated multiple times according to a parameter n_{rep}.

Algorithm 14 *createTasks*

Require: target task $\mathcal{T}_t$, repetition parameter n_{rep}.

1: $\boldsymbol{\mathcal{T}} \leftarrow \emptyset$
2: $\boldsymbol{F}_{simple} \leftarrow simplify(\boldsymbol{O}^{\mathcal{T}_t}_{C_{Env}})$
3: $C_{min} \leftarrow \arg\min_{C_i \in \boldsymbol{C} - C_{Env}} |\boldsymbol{O}^{\mathcal{T}_t}_{C_i}|$
4: $o_{min} \leftarrow |\boldsymbol{O}^{\mathcal{T}_t}_{C_{min}}|$
5: **for** n_{rep} times **do**
6: **for** $\forall i \in \{0, \ldots, o_{min}\}$ **do**
7: Draw F from $\boldsymbol{F}_{simple}$
8: $\boldsymbol{O} \leftarrow$ Draw i objects from $\boldsymbol{O}^{\mathcal{T}_t}_{C_{min}}$
9: **for** $\forall C_i \in \boldsymbol{C} - C_{Env}$ **do**
10: Draw q from $\{0, \ldots, |\boldsymbol{O}^{\mathcal{T}_t}_{C_i}|\}$
11: $\boldsymbol{O} \leftarrow \boldsymbol{O} \cup$ Draw q objects from $\boldsymbol{O}^{\mathcal{T}_t}_{C_i}$
12: $\boldsymbol{O} \leftarrow \boldsymbol{O} \cup F$
13: $S_0 \leftarrow initState(\boldsymbol{O}, \mathcal{T}_t)$
14: $A \leftarrow actionSpace(\boldsymbol{O}, \mathcal{T}_t, A^{\mathcal{T}_t})$
15: $T \leftarrow transitionFunction(T^{\mathcal{T}_t})$
16: $R \leftarrow rewardFunction(R^{\mathcal{T}_t})$
17: $\boldsymbol{\mathcal{T}} \leftarrow \boldsymbol{\mathcal{T}} \cup \langle \boldsymbol{C}, \boldsymbol{O}, S_0, A, T, R \rangle$
18: **return** $\boldsymbol{\mathcal{T}}$

This procedure can be used to generate a set of source tasks when a human is unavailable or unwilling to provide it. However, this procedure is not valid for all possible domains, because we change the number of objects without necessarily having knowledge about the transition function. When using this procedure, the designer must ensure that solvable tasks are generated. Thus, it might be necessary to set a minimum number of objects of a given class, to create the initial state in a domain-dependent

[5] If transition and reward functions are unknown, the environment will implicitly provide samples of those functions to the agent.

way, or to add a final human-guided step to reject unsolvable tasks (which is still much easier than hand-coding the entire set).

In the next section we present our experimental evaluation.

5.3.3 Experimental Evaluation

We have chosen two domains for our experimental evaluation. The *Gridworld* domain, as proposed by Sevtlik *et al.* (SVETLIK et al., 2017), and the *Half Field Offense* (HFO) (HAUSKNECHT et al., 2016) environment. The former shows the performance of the proposed method in a domain easy to implement and to gather results, while the latter shows the robustness of the method in a very challenging multiagent task with continuous observations[6]. We evaluate our proposal both with manually given (*OO-Given*) and generated (*OO-Generated*) source tasks, comparing it with Svetlik's (SVETLIK et al., 2017) *Curriculum* generation procedure (hereafter named *Svetlik*) and the regular learning without a *Curriculum* (Q-Learning for *Gridworld* and SARSA for HFO - hereafter called as *No Curriculum*). The TL procedure is carried out as defined in Equation (5.12) for our proposal and through transfer of value functions (TAYLOR; STONE; LIU, 2007) for *Svetlik*.

Gridworld

Figure 5.14 illustrates our *Gridworld*. Each cell in the grid may have one of the following objects or be empty: *fire*, *pit*, or *treasure*. The agent can move in four cardinal directions $A = \{North, South, East, West\}$, and the task is solved when the agent collects the treasure. The *Object-Oriented* description of the task has the classes $C = \{Pit, Fire, Treasure, Env\}$, where the environment attributes are the sizes of the grid $Att(Env) = \{sizeX, sizeY\}$ and the remaining classes have x and y attributes. The position attributes are observed in regard to the distance between the agent and the object, rather than the absolute position. The rewards are defined as in (SVETLIK et al., 2017). At each of every executed step, the agent observes one of the following rewards: (i) **+200** for collecting the treasure; (ii) **-250** for getting next to a fire (4-neighborhood); (iii) **-500** for getting into a fire; (iv) **-2500** for falling into a pit; and (v) **-1** if nothing else happened.

The target task is illustrated in Figure 5.14, and contains 8 pits, 11 fires, and 1

[6]Notice that, in the current work, we assume that the sequence of tasks is defined by a single agent. In case those are multiagent tasks, additional agents might join the learning agent in the tasks selected by it when required, but they cannot interfere with the *Curriculum* generation and use.

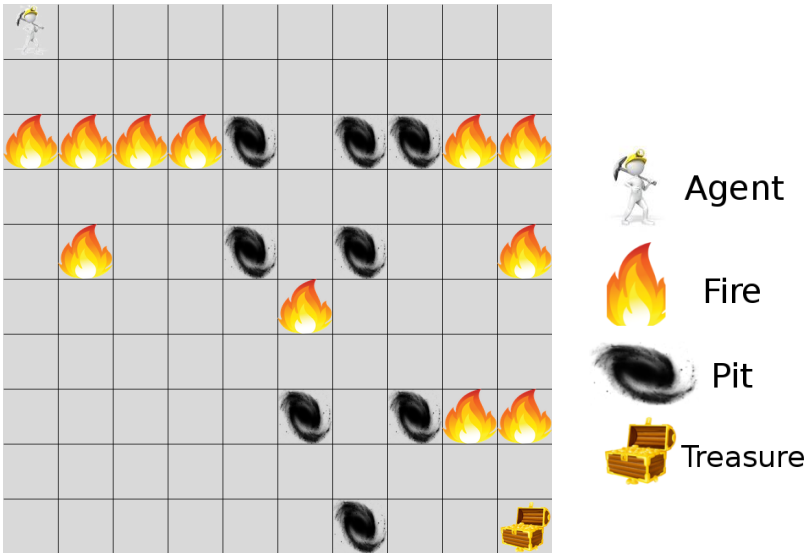


Figure 5.14. An illustration of the target task in Gridworld.

treasure. As in (SVETLIK et al., 2017), we generate a set of source tasks by reducing the number of objects and/or reducing the size of the grid, defining $|\mathcal{T}| = 18$ source tasks. During learning, all source tasks given by the agent *Curriculum* were executed until the agent achieved the same cumulative reward for 2 consecutive episodes, or 30 learning episodes were carried out. Episodes start in the task initial state (e.g., the configuration shown in Figure 5.14) and end when the agent captures the treasure or falls into a pit.

The comparison metric here is the cumulative reward when trying to solve the target task. The threshold parameter for *Curriculum* generation was set $\epsilon = 1$, $n_{traj} = 3$, and $n_{steps} = 3$ for our proposal and $\epsilon = 15$ for *Svetlik*[7]. The task generation parameters for our proposal are $\epsilon = 0.5$, $n_{rep} = 2$, $n_{traj} = 2$, and $n_{steps} = 2$.

HFO

In our setting for this experiment, a learning agent has to score a goal against a team of two highly specialized defensive agents.

A learning episode starts with all agents initiated in a random position (defensive agents always near the goal), and the ball possession with the offensive agent. One opponent plays the role of goalkeeper, while the other is a defender. The episode ends when the agent scores a goal, a defensive agents captures the ball, the ball leaves the field, or after 200 game frames. In spite of only having two available actions when carrying the ball, this task is still very hard to solve, as a reward is received only

[7]The parameters were defined in preliminary experiments.

when the episode ends and the agent scores a goal only when shooting at a propitious moment. The Helios strategy (when also playing as the offensive agent) scores in roughly 30% of the attempts, while a random agent has a score of roughly 3% of the attempts. We make use of the following observations[8] of the current state for this experiment, normalized in the range $[-1, 1]$:

- **Goal Proximity** – the distance from agent to goal center;
- **Goal Angle** – the angle from agent to goal center;
- **Goal Opening** – the largest angle from agent to goal with no blocking agent;
- **Opponent Proximity** – the distance between the agent and the nearest opponent.

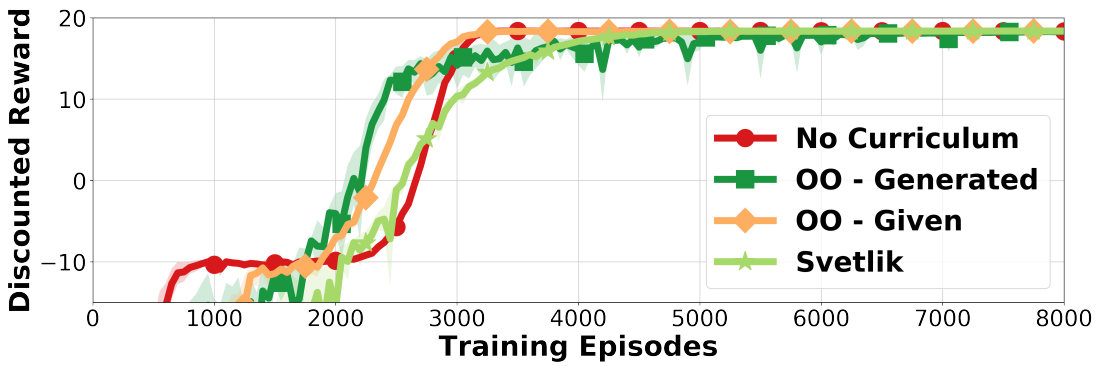
The observations are discretized by Tile Coding (SHERSTOV; STONE, 2005) configured with 5 tiles of size 48 and equally spread over the range. The *Object-Oriented* description of the task has the classes $C = \{Agent, Opponent, Env\}$, in which the environment attribute is the average initial distance between the offensive agent and the goal $Att(Env) = \{dist\}$, *Agent* has attributes corresponding to the aforementioned observations, and *Opponent* has a single attribute specifying the agent strategy (*Helios* or *Base*). We generated $|\mathcal{T}| = 8$ tasks, in which we vary the initial distance between the teams and the number and strategy of opponents. The source tasks are executed until one of the following conditions is true: (i) the agent scores 80% of the attempts in the last 20 episodes; (ii) no goal is scored in the last 50 episodes; or (iii) after 500 learning episodes.

The threshold parameter for *Curriculum* generation was set $\epsilon = 0.5$, $n_{traj} = 3$, and $n_{steps} = 3$ for our proposal and $\epsilon = 0.75$ for *Svetlik*. The task generation parameters for our proposal are $\epsilon = 0.5$, $n_{rep} = 20$, $n_{traj} = 3$, and $n_{steps} = 3$.

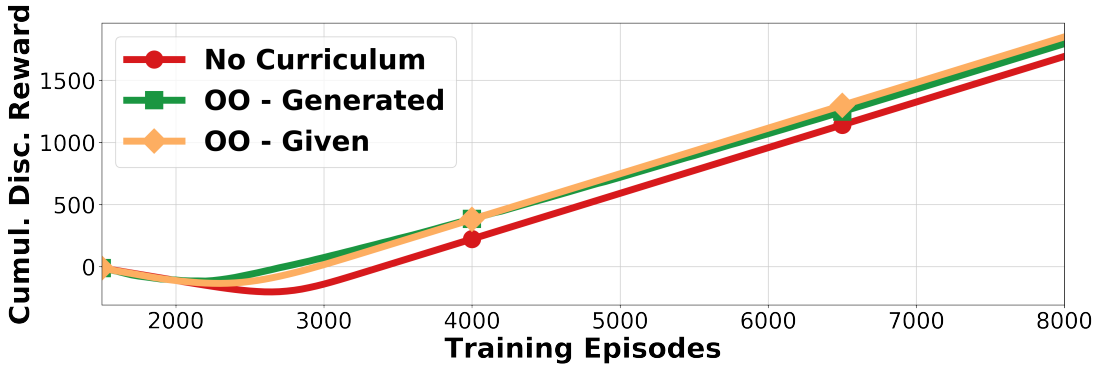
5.3.4 Results Gridworld

Figure 5.15a shows the performance observed when solving the target task in the *Gridworld* domain. Although *Svetlik* presents very good results in (SVETLIK et al., 2017), we found out that their proposal is quite sensitive to parameters and to the provided source tasks. The performance shown in our experiments is the best result observed after trying several parameters, but it is still a little worse than *No Curriculum* if the steps in source tasks are also taken into account. In turn, both *OO-Generated* and *OO-Given* achieved positive results when compared to *No Curriculum*. Both of

[8] A subset of the full list of observations that can be used.



(a)



(b)

Figure 5.15. The average discounted rewards observed in 2,000 repetitions of the experiment in the *Gridworld* domain. (a) refers to the average performance during learning; and (b) to the cumulative rewards starting from step 1,500. Steps used to learn source tasks are also considered in the graph. The shaded area is the 95% confidence interval.

them achieved a slightly worse performance than *No Curriculum* until 1,500 learning steps. Then, our proposal learns how to avoid the negative rewards faster, achieving better results than *No Curriculum* between 1,700 and 3,000 learning steps. Then, *OO-Given* achieves the optimal policy few steps before *No Curriculum* while *OO-Generated* takes a little longer, but *OO-Generated* has already settled in a very good performance (near the optimal) by then. Figure 5.15b shows the accumulated reward in the target task starting from 1,500 steps for both configurations of our proposal and *No Curriculum*. All the three algorithms have similar negative cumulative reward up to roughly 2,300 learning steps, after which the improvement in the learning process when using our proposal becomes clear. This shows that even though fewer steps are needed in the target task, the performance achieved in training is better.

Although Figure 5.15a assumes that learning steps have the same difficulty for the agent in both the source and target tasks, for some applications learning in source tasks may be easier (for example, if source tasks are learned in a virtual simulator and the target task is in the real world (HANNA; STONE, 2017)). Therefore, Figure 5.16

shows the results if the learning steps in source tasks are not considered. The difference is not dramatic for our approach (but still slightly better than in Figure 5.15a), as the source tasks are learned very fast. However, for *Svetlik* this represents a huge "speed-up". Now *Svetlik* learns faster than *No Curriculum* how to avoid negative rewards and it is even a little better than our proposal, despite taking longer than *OO-Given* to converge to the optimal policy. The results here show that both Svetlik's and ours proposals might be useful if steps in the source task are less costly than in the target task. Notice that the transfer potential metric is easier to compute with our proposal if an object-oriented description is available, though.

5.3.5 Results HFO

Figure 5.17a shows the goal percentage when solving the task for all algorithms. As the agents have a challenging task to solve, it is very hard to improve performance. At the end of training, all algorithms score goals around 20% of the attempts. This is a good performance, as the *Helios* team scores roughly in 30% of the attempts and it uses several hand-coded specialized strategies.

None of the *Curriculum Learning* approaches can achieve a better performance than *No Curriculum* at the end of learning, but all *Curriculum* approaches learn in the target task for less time. *OO-Given* starts learning in the target task around episode 6,000, and quickly reaches *No Curriculum* performance. *OO-Generated* takes longer to start learning in the target task, but the performance achieved is already similar to *No Curriculum* as soon as it starts learning in the target task. *Svetlik* also takes very long to start learning in the target task, which happens at roughly episode 52,000. After starting learning in the target task, *Svetlik* takes longer than our proposal to achieve *No Curriculum* performance, getting the same performance than the other algorithms at

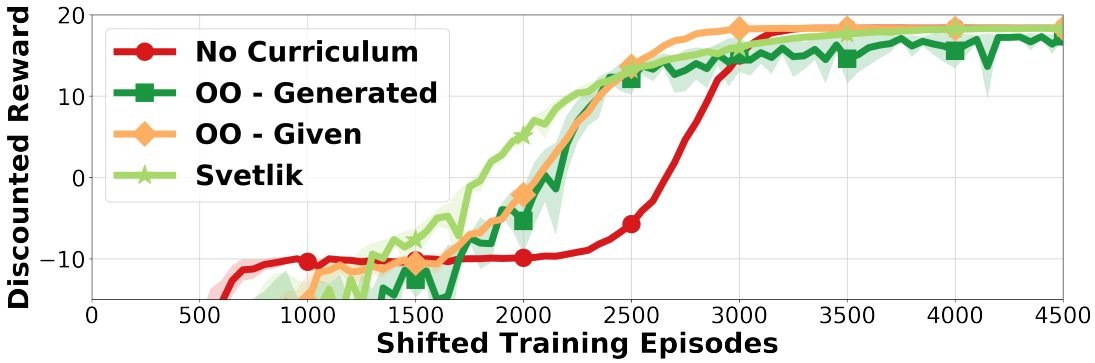
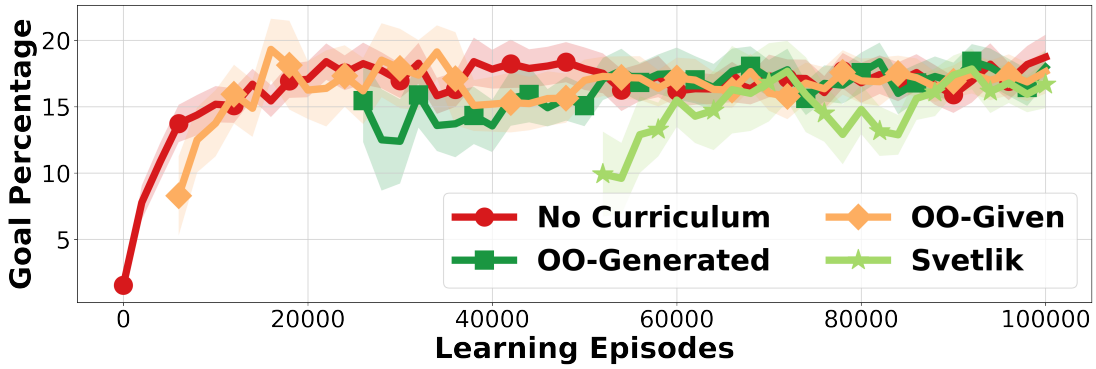
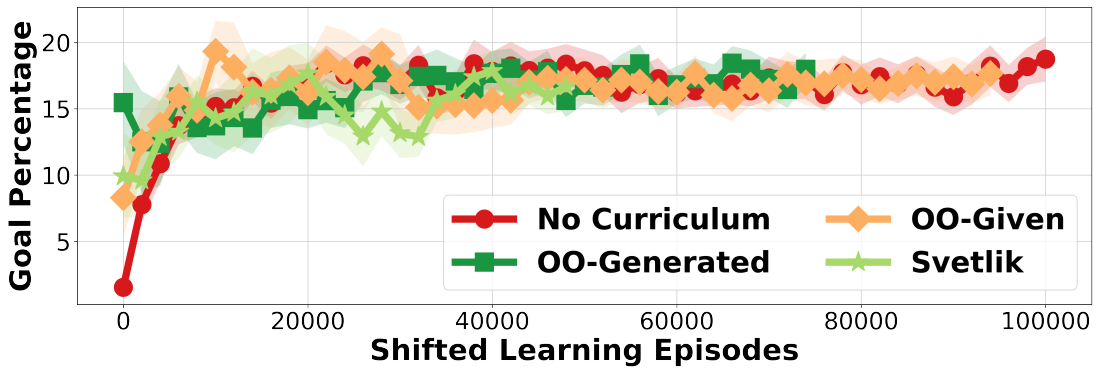


Figure 5.16. The average discounted rewards disregarding steps carried out in source tasks in the *Gridworld* domain. The shaded area is the 95% confidence interval.

roughly episode 70, 000.



(a)



(b)

Figure 5.17. The average percentage of goals observed in 50 repetitions of the experiment in the *HFO* domain when the steps used to learn source tasks are: (a) considered; and (b) not considered. The shaded area is the 95% confidence interval.

Figure 5.17b more clearly shows that our proposal achieved a good performance faster than *No Curriculum* when taking into account only steps in the target task. *Svetlik* was also slightly better than *No Curriculum* but worse than both settings of our proposal. The number of cumulative goals scored by each approach (not shown graphically here) indicates that *Svetlik* achieves the same cumulative performance as *No Curriculum*, while both *OO-Given* and *OO-Generated* achieve a slightly better performance, thus making our proposal a better choice for this domain.

5.3.6 Discussion

In both experiments *OO-Given* and *OO-Generated* presented a better performance than *Svetlik* and *No Curriculum* in general. In the *Gridworld* domain our proposal achieved a good performance clearly faster than not using *Curriculum*, presenting advantages in both performance and number of steps executed in the target task. In the *HFO* domain, our proposal achieved a similar performance than not using *Curriculum*,

but still required fewer steps training in the target task. Our experiments also show that the performance of our proposal when generating tasks (*OO-Generated*) is consistent and satisfactory, as in both domains the use of generated tasks achieved a competitive performance when compared to the manually generated set of tasks, which requires more manual intervention and domain-knowledge.

Chapter 6

Conclusions

Despite the impressive successes achieved by Reinforcement Learning in the past years, real-world complex multiagent applications are still yet to come. Scaling techniques that have been very successful in single-agent domains has been shown to be a hard task. Multiple agents actuating in the environment bring many new prospective domains to be solved with RL and a promise of efficient, parallelizable, and fault-proof systems. However, the explosion of the state-action space together with the non-stationarity of the learning problem makes simple adaptations of single-agent algorithms inefficient for all but the most simple instances of problems. Knowledge reuse comes into play as a very important asset to scale multiagent RL. If agents are able to consistently reuse past knowledge and to communicate gathered knowledge between them, solving tasks as a team become much easier and faster. However, there are a miryad of possible multiagent scenarios. The system can be composed of cooperative, adversarial, or self-interested agents; the actions of other agents might be visible or not; agents might be homogeneous or heterogeneous; etc. Many of those scenarios gave rise to its own subcommunity, and many of the proposed methods of the literature are very specific to its scenario with very little inter-operability with other subcommunities' work, which hampers scaling up those methods to real-world applications. We argue that our community should have a more unified view, making the methods more flexible and inter-operable.

Our work aimed at unifying the ideas of the contemporary literature through the introduction of flexible knowledge reuse methods that could be combined. Throughout this thesis, we propose ways to reuse knowledge from both previous solved tasks and communication with other agents. *Ad Hoc Advising* (Chapter 4) represents the state-of-the-art for transferring knowledge through action advice. This method is our answer to the question introduced in the beginning of this thesis: *when and how to trans-*

mit knowledge efficiently?, enabling the participation of knowledge exchange across agents. With small modifications, our method is applicable across different scenarios, and can be used to transfer knowledge from humans or automated agents following any learning algorithm alike.

MOO-MDP was proposed as a way to facilitate knowledge abstraction (Chapter 5), and as our answer to the questions *how to abstract acquired knowledge, in order to better generalize and reuse it?*. MOO-MDP is easy to specify and understand, which makes it potentially helpful for devising ways of including laypeople on the loop while facilitating knowledge reuse. We also propose ways of using MOO-MDP to assist the reuse of knowledge through object-oriented inter-task mappings and *Curriculum Learning*. Our proposed methods solve complementary knowledge reuse scenarios, and can be used both separately with state-of-the-art performance in their intended scenarios or combined to provide both Inter- and Intra-Agent transfer. While each of our contributed methods opens up several possible lines of further research, we hope our efforts will help the community to bridge the gap between the current state-of-the-art in the area and complex real-world applications of multiagent RL. Both *Ad Hoc Advising* and tranfer based on MOO-MDP answer the question: *How to represent knowledge?* in complementary scenarios.

In the next Chapter we discuss potential research avenues following or related to our work. We expect our main long-term contribution to the field to be our vision of proposing TL methods that spam across multiple types of knowledge reuse, which we expect to be a very profitable line of research.

Chapter 7

Further Work

TL for MAS has already played an important role towards scaling RL to more complex problems. However, the field is still fragmented in communities developing methods with little interconnection and flexibility. Our work described in this thesis provided an analysis of the literature and framed a unified view of the main developments of the state of the art, as well as proposed novel and flexible methods on the area. Our work opens up several avenues to be explored in further work towards efficient, flexible, and easily applicable TL algorithms for complex MAS. We discuss prominent lines of research that will require further investigations in the years to come, both in the form of extension of our methods described in this thesis and as open questions that we not addressed completely in this thesis.

- **Curriculum Learning in MAS** - The ideas introduced by *Curriculum* Learning techniques can greatly benefit RL agents in MAS. Our work described in Sections 5.3.1 improved the state of the art in the area. However, many research directions are still open as this subarea is still young.

  The *Transfer of Curriculum* is perhaps the most relevant open problem for this thesis (which has as early techniques the ones cited in Section 3.4.6). We expect that future methods will be able to adapt an existent *Curriculum* to a different agent, tailoring it according to the agent's unique capabilities and particularities. This is a very challenging goal, as the information about the target task is usually scarce for RL agents, which is further complicated in MAS where the strategy of other agents is unknown.

  As shown in Section 5.3.1, pruning a *Curriculum* before executing the tasks might be required depending on how the tasks were generated. Although our solution based on a *Random Walk* was effective in practice, finding *principled*

ways to prune a *Curriculum* is required for achieving performance guarantees.

- **Benchmarks for Transfer in MAS** - Being able to numerically compare two algorithms is very useful for the AI community, as a proper metric is a very impartial and clear method for analyzing the relative performance when solving a given problem. However, no single TL metric is sufficient to have a clear picture of the performance, and even all of the currently-used ones together might be inefficient for MAS techniques.

  For understanding how evaluations of TL methods might be misleading, consider the fictitious performances in Figure 7.1a. *Alg1* seems to have a better

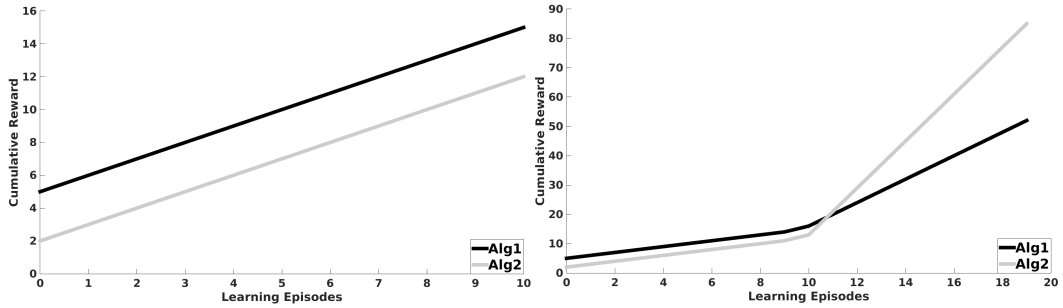


Figure 7.1. Fictitious performance comparison of two algorithms. (a) Running 10 episodes; (b) Running 20 episodes.

performance in this graph, as both jumpstart and end of learning performances are higher than the one observed for *Alg2*. Now suppose that we train the agents for longer and the result is observed in Figure 7.1b. Notice that now *Alg2* seems to be better, because in spite of the lower jumpstart the end of learning performance is much higher. Therefore, choosing among *Alg1* and *Alg2* is simply arbitrary if the number of learning episodes is chosen without care. This analysis is further complicated if we consider costs of communication for transfer of knowledge between agents. If one method has a better performance but transfers a much higher number of messages, which of them can be considered as the best one? There is no single answer to this question, and performance comparisons must be carried out in a very domain-specific and subjective way. The difficulty of comparing methods and the differences in assumptions across TL methods are partially responsible for the lack of comparisons between methods in a significant portion of the literature.

Therefore, one of the most important open problems in the TL for MAS area is the development of proper comparison methods. In addition to the introduction of new metrics, the definition of standard benchmarks for comparing algorithms

is required, similarly as proposed by Vamplew *et al.* (2011) for *Multiobjective* RL. Benchmark problems would allow quickly deploying a new TL algorithm and comparing it to similar previous methods.

- **Knowledge Reuse for Ad Hoc Teams** - In *Ad Hoc* teams (STONE et al., 2010), agents have to solve collaborative tasks with previously-unknown teammates, which means that no predefined communication protocol is available when starting learning. For this reason, learning to coordinate with each new teammate takes some time, and exchanging information is often unfeasible for the lack of a common protocol.

  However, it might be possible to reuse knowledge gathered from playing with another similar teammate. For this purpose, the learning agent would need to be able to estimate the similarity between the current and past teammates, and also to properly reuse the previous knowledge.

  Performing knowledge reuse would be especially challenging if teammates do not follow a fixed strategy and are also learning. In this case, computing similarities and reusing knowledge could be harder, but would also make the RL agent more flexible and robust to sudden changes in the system.

- **End-to-End Multiagent Transfer Frameworks** - Although usually enabling the RL agent to learn autonomously after deployed in the environment, the current state-of-the-art methods still require considerable human effort and expertise in order to be applicable. Most of the methods require parameters that must be specified in a domain-specific manner, and communication and knowledge transfer protocols are usually implicitly hard-coded in the agent's implementation.

  All those definitions, that are painstakingly optimized for each domain, hurt the applicability of the methods, and for this reason, most of the successful applications of the papers here discussed are in controlled environments.

  In order to create more robust agents, the community needs to create a new line of research on the development of end-to-end transfer learners. The main goal of this line would be building agents able to autonomously create, optimize, and use protocols for negotiating knowledge with other agents, tune their parameters, and find the best way to combine internal knowledge with received information. The initial works on autonomously learned communication (SUKHBAATAR; SZLAM; FERGUS, 2016; FOERSTER et al., 2016) are related to this challenge and might be seen as a first step towards autonomously learning transfer protocols.

- **Transfer for Deep Multiagent Reinforcement Learning** - Deep RL has achieved many impressive successes in recent years (SILVER et al., 2016), especially for problems where it is hard to extract useful engineered features from raw sensor readings (such as images) and is now starting to be employed in MAS (CASTANEDA, 2016; GUPTA; EGOROV; KOCHENDERFER, 2017). Even though all of the ideas in discussed here could be used to some extent for Deep RL, most of the methods are not directly usable in agents learning with Deep Networks. Therefore, the adaptation of transfer methods for Deep multiagent RL is a promising area of research.

  The preliminary investigations discussed in Section 3.4.8 showed that Deep RL agents also benefit from reuse of knowledge, but the effects of negative transfer could be catastrophic if unprincipled transfer is performed (GLATT; SILVA; COSTA, 2016; DU et al., 2016). Those initial efforts in this topic still need further investigations for scaling Deep RL to complex MAS applications.

- **Integrated Inter-Agent and Intra-Agent Transfer** - As shown in our analysis of the literature, Inter- and Intra-Agent transfer are very rarely integrated, and most of the methods focus on one of them alone. However, humans routinely combine both types of knowledge reuse, and it is most likely that agents integrated into our domestic and corporate environments will have to consistently implement both types of knowledge reuse.

  This challenge was highlighted and outlined by our research, and implemented to a very limited extent by some previous methods (MADDEN; HOWLEY, 2004; WANG et al., 2016). However, many problems inherent from combining knowledge from different sources remain unsolved. Imagine that a teacher is willing to provide demonstrations to the student, while a mentor is ready to be observed, and a previously learned policy from a similar problem is available. Should the agent follow the demonstration, the previous knowledge, or try to imitate the mentor? Would it be possible to combine knowledge from all those sources? Those and many other questions that would be unveiled by a deeper investigation are not answered by the current literature.

- **Human-focused Multiagent Transfer Learning** - At first sight, transferring knowledge from a human seems to be easy. Humans are very good at abstracting knowledge and carry years of previous experiences from which automated agents could profit. However, humans have a different set of sensors, reaction times, and have a limited capability of focusing on a task. Those differences might lead the human to provide useless or even inconsistent knowledge in the

point of view of the agent. Those problems are aggravated when knowledge is transferred from non-expert humans, which might not understand that automated agents follow a different reasoning process and do not have the amount of previous knowledge that humans have. Still, as AI experts represent a tiny portion of the human population, laypeople will be required to instruct automated agents in the near future. Therefore, even though already explored by a good portion of the *Inter-Agent* transfer literature, there is still much to be improved on human-focused transfer. As a way to facilitate the participation of non-experts on agent training, additional investigations are needed to develop appropriate methods for translating common human languages into instructions that are useful for RL agents. Also, it is desired that the methods take advantage of the psychological bias to incentive humans to give good instructions, for example, reducing the agent's actuation speed to indicate that the agent has a high uncertainty for a given state, as Peng *et al.* (2016a) do.

The current literature also neglects issues related to human reaction times. Simulated environments are often paused for receiving human feedback, which is impossible for many tasks in the real world. This is especially important for robotic tasks, as a delayed corrective feedback could lead to a harmful movement that cannot be interrupted in time for avoiding a collision.

Transfer methods are also desired to be easily readable and understandable for humans (RAMAKRISHNAN; NARASIMHAN; SHAH, 2016), as "black-box" methods should not be employed in critical applications.

- **Cloud Knowledge Bases** - Kono *et al.* (2014) introduced the innovative idea of having an ontology available in the web for translating the agent's individual capabilities into a more abstract set of skills, which would enable, e.g., the transfer of knowledge between two robots manufactured by different companies.

  Their idea could be extended for building *web-based knowledge bases*, that could be used by agents as a common knowledge repository to extract and add knowledge (playing the role of the internet in the modern human life). The idea is illustrated in Figure 7.2.

  When a newly-built agent is joining a system or starts learning in a task, it will make use of an ontology to translate its own sensor readings and low-level actions into abstract states and skills that are common to all similar robots. Then, an abstract policy could be used to bootstrap learning, or alternatively, the base could inform the IPs of similar robots with which the agent could share knowledge.

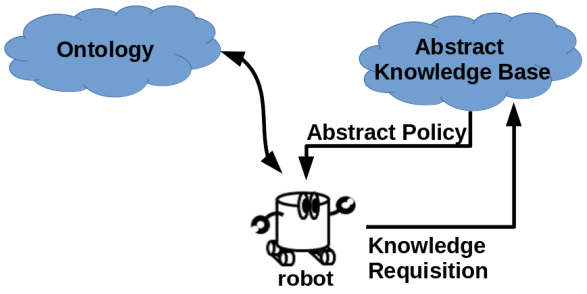


Figure 7.2. Illustration of how a web-based knowledge base could be built. The agent accesses an ontology base for defining abstract state and action spaces. Then, an online knowledge base provides knowledge to the agent. Inspired by Kono *et al.*'s illustration (KONO et al., 2014)

The major challenge of implementing such architecture would be preventing agents from adding incorrect or malicious knowledge into the knowledge base, which is related to the next challenge.

- **Security** - Nearly all of the current literature assumes that agents involved in knowledge reuse interactions are benevolent, i.e., they will never communicate wrong or inconsistent knowledge on purpose, but this is rarely the case for real-world applications. In fact, in many of the methods discussed throughout this thesis, the agent transferring knowledge is assumed to have perfect knowledge of the task and to know the optimal policy, while a more realistic assumption would be that a reasonable policy can be estimated through TL but the agent still needs to explore the environment for learning the optimal policy.

  As MAS are progressively scaled for real-world applications, RL researchers need also to be concerned about security. If agents base their exploration on knowledge received from communication, malicious agents could be possibly able to transfer inconsistent or incorrect knowledge for affecting the learning agent's performance. For robotic systems or autonomous cars, a malicious communication could cause serious damages to very expensive equipment. If the agent is deployed in uncontrolled environments, it is also possible to cause damage to people, other agents, or buildings.

  Therefore, the development of transfer methods that are robust to interference is imperative. Reward Shaping methods (Section 3.4.3) have a solid theoretical background and are one of the few families of transfer algorithms proved to converge to the optimal policy even when malicious transfer is performed. For this reason, those methods might have an important role towards applications in which security is critical. Another possible step towards secure methods could be the development of argumentation protocols, in which agents will have to provide arguments explaining why the transferred knowledge is reliable and correct.

Deploying security into knowledge-transferring agents is a very challenging but necessary research goal.

- **Inverse Reinforcement Learning for Enforcing Cooperation** - Although IRL is most commonly used only for defining a reward function through demonstrations to learning agents, the recent trend on using such techniques for multiagent tasks opens up new research opportunities (LIN; BELING; COGILL, 2018).

  Natarajan *et al.* (2010) solve a multiagent cooperative problem by using IRL for learning the reward function of all the agents in the environment (which have a locally optimal reward). Then, a centralizer agent enforces the agents to change their actions to the expected global optimal policy, where they were not doing it already. Even though having a centralized agent dictating the joint actions is unfeasible to most domains, this agent could oversee the actuation of the other agents and introduce additional rewards as an incentive to cooperation. Such framework, if successfully developed, could be a possible solution for domains in which an institution wants to incentivize cooperation between self-interested agents (such as some of the techniques studied by the area of *Organization of MAS* (ARGENTE; JULIAN; BOTTI, 2006)).

- **Scaling Up Ad Hoc Advising** - As discussed and explored in Section 4.2, proposing more flexible and generally-applicable confidence functions is the main step towards making Ad Hoc Advising widely compatible with most RL base algorithms.

  We here consider that the set of potential advisors is given and fixed. However, for some domains the set of reachable agents might be initially unknown and/or dynamic. Choosing the advisors dynamically as in Garant *et al.*'s (2015) would be more realistic in uncontrolled environments.

  The most challenging (yet very relevant) improvement would be extending Ad Hoc Advising to scenarios where agents cannot be assumed to be benevolent. Accounting for a potentially compromised or malicious advisor would perhaps require the development of trust mechanisms to estimate the quality of advisors.

- **Extensions to the Multiagent Object-Oriented Representation** - Although we have proposed an extension of OO-MDP to the multiagent case and successfully used it to perform TL (Chapter 5), our proposal can still be extended to work in additional more complicated scenarios. The first step would be developing algorithms for MOO-MDPs where DOO-Q is not applicable, such as general-sum games and continuous domains. Those algorithms could also potentially

explore different exploration strategies that cannot be applied with DOO-Q, such as optimistic exploration (DIUK; COHEN; LITTMAN, 2008).

MOO-MDPs could also be extended to model partial observably (MELO; VELOSO, 2011), which would enable distributed solutions that reason over observations rather than assuming that the full state is available.

REFERENCES

ABEL, D.; SALVATIER, J.; STUHLMÜLLER, A.; EVANS, O. Agent-Agnostic Human-in-the-Loop Reinforcement Learning. In: **Proceedings of the NIPS Future of Interactive Learning Machines Workshop**. 2016.

AKIYAMA, H. **Helios team base code. <https://osdn.jp/projects/rctools/>**. 2012.

AMIR, O.; KAMAR, E.; KOLOBOV, A.; GROSZ, B. Interactive Teaching Strategies for Agent Training. In: **Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)**. 2016. p. 804–811.

ARGALL, B. D.; CHERNOVA, S.; VELOSO, M.; BROWNING, B. A Survey of Robot Learning from Demonstration. **Robotics and Autonomous Systems**, v. 57, n. 5, p. 469 – 483, 2009. ISSN 0921-8890.

ARGENTE, E.; JULIAN, V.; BOTTI, V. Multi-Agent System Development Based on Organizations. **Electronic Notes in Theoretical Computer Science**, v. 150, n. 3, p. 55 – 71, 2006. ISSN 1571-0661.

BANERJEE, B.; STONE, P. General Game Learning using Knowledge Transfer. In: **Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)**. 2007. p. 672–677.

BANERJEE, B.; TAYLOR, M. E. Coordination Confidence based Human-Multi-Agent Transfer Learning for Collaborative Teams. In: **AAMAS Adaptive Learning Agents (ALA) Workshop**. 2018.

BARRETT, S.; STONE, P. Cooperating with Unknown Teammates in Complex Domains: A Robot Soccer Case Study of Ad Hoc Teamwork. In: **Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI)**. 2015. p. 2010–2016.

BIANCHI, R.; ROS, R.; MANTARAS, R. L. de. Improving Reinforcement Learning by Using Case Based Heuristics. **Case-Based Reasoning Research and Development**, Springer, p. 75–89, 2009.

BIANCHI, R. A. C.; MARTINS, M. F.; RIBEIRO, C. H. C.; COSTA, A. H. R. Heuristically-Accelerated Multiagent Reinforcement Learning. **IEEE Transactions on Cybernetics**, v. 44, n. 2, p. 252 – 265, 2014.

BOUTSIOUKIS, G.; PARTALAS, I.; VLAHAVAS, I. Transfer Learning in Multi-agent Reinforcement Learning Domains. In: **Proceedings of the 9th European Workshop on Reinforcement Learning**. 2011. Available in: <http://ewrl.files.wordpress.com/2011/08/ewrl2011_submission_19.pdf>.

BOWLING, M.; VELOSO, M. An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning. **Technical Report, Computer Science Department, Carnegie Mellon University**, 2000.

BRAYLAN, A.; MIIKKULAINEN, R. Object-Model Transfer in the General Video Game Domain. In: **Proceedings of the 12th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)**. 2016. p. 136–142.

BRYS, T.; HARUTYUNYAN, A.; SUAY, H. B.; CHERNOVA, S.; TAYLOR, M. E.; NOWÉ, A. Reinforcement Learning from Demonstration through Shaping. In: **Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)**. 2015. p. 3352–3358.

BUSONIU, L.; BABUSKA, R.; SCHUTTER, B. D. A Comprehensive Survey of Multiagent Reinforcement Learning. **IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews**, v. 38, n. 2, p. 156–172, 2008. ISSN 1094-6977.

CASTANEDA, A. O. **Deep Reinforcement Learning Variants of Multi-Agent Learning Algorithms**. Thesis (Doutorado) — University of Edinburgh, 2016.

CHERNOVA, S.; VELOSO, M. Interactive Policy Learning through Confidence-Based Autonomy. **Journal of Artificial Intelligence Research (JAIR)**, v. 34, n. 1, p. 1–25, 2009.

CROONENBORGHS, T.; TUYLS, K.; RAMON, J.; BRUYNOOGHE, M. Multi-agent Relational Reinforcement Learning. In: **Learning and Adaption in Multi-Agent Systems**. 2005. p. 192–206.

CRUZ, G. V.; DU, Y.; TAYLOR, M. E. Pre-training Neural Networks with Human Demonstrations for Deep Reinforcement Learning. **The Knowledge Engineering Review**, v. 34, p. e10, 2019.

CUI, Y.; NIEKUM, S. Active Reward Learning from Critiques. In: **IEEE International Conference on Robotics and Automation (ICRA)**. 2018. p. 6907–6914.

DENIL, M.; AGRAWAL, P.; KULKARNI, T. D.; EREZ, T.; BATTAGLIA, P.; FREITAS, N. de. Learning to Perform Physics Experiments via Deep Reinforcement Learning. In: **Proceedings of the 5th International Conference on Learning Representations (ICLR)**. 2017.

DEVIN, C.; GUPTA, A.; DARRELL, T.; ABBEEL, P.; LEVINE, S. Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer. In: **IEEE International Conference on Robotics and Automation (ICRA)**. 2017. p. 2169–2176.

DEVLIN, S.; KUDENKO, D. Theoretical Considerations of Potential-Based Reward Shaping for Multi-Agent Systems. In: **The 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2011. p. 225–232.

DEVLIN, S.; YLINIEMI, L.; KUDENKO, D.; TUMER, K. Potential-Based Difference Rewards for Multiagent Reinforcement Learning. In: **Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2014. p. 165–172.

DIDI, S.; NITSCHKE, G. Multi-agent Behavior-Based Policy Transfer. In: SQUILLERO, G.; BURELLI, P. (Ed.). **Proceedings of the 19th European Conference on Applications of Evolutionary Computation (EvoApplications)**. 2016. p. 181–197. ISBN 978-3-319-31153-1. Available in: <https://doi.org/10.1007/978-3-319-31153-1_13>.

DIETTERICH, T. G. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. **Journal of Artificial Intelligence Research (JAIR)**, v. 13, p. 227–303, 2000.

DIUK, C. **An Object-Oriented Representation for Efficient Reinforcement Learning**. Thesis (Doutorado) — Rutgers University, 2009.

DIUK, C.; COHEN, A.; LITTMAN, M. L. An Object-oriented Representation for Efficient Reinforcement Learning. In: **Proceedings of the 26th International Conference on Machine Learning (ICML)**. 2008. p. 240–247. ISBN 978-1-60558-205-4.

DU, Y.; CRUZ, G. V. de la; IRWIN, J.; TAYLOR, M. E. Initial Progress in Transfer for Deep Reinforcement Learning Algorithms. In: **Proceedings of Deep Reinforcement Learning: Frontiers and Challenges Workshop at IJCAI**. 2016.

FACHANTIDIS, A.; TAYLOR, M. E.; VLAHAVAS, I. Learning to Teach Reinforcement Learning Agents. **Machine Learning and Knowledge Extraction**, v. 1, n. 1, 2018. ISSN 2504-4990.

FERNANDEZ, F.; VELOSO, M. Probabilistic Policy Reuse in a Reinforcement Learning Agent. In: **Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2006. p. 720–727. ISBN 1-59593-303-4.

FLORENSA, C.; HELD, D.; WULFMEIER, M.; ZHANG, M.; ABBEEL, P. Reverse Curriculum Generation for Reinforcement Learning. In: LEVINE, S.; VANHOUCKE, V.; GOLDBERG, K. (Ed.). **Proceedings of the 1st Conference on Robot Learning (CoRL)**. : PMLR, 2017. (Proceedings of Machine Learning Research, v. 78), p. 482–495.

FOERSTER, J. N.; ASSAEL, Y. M.; FREITAS, N. de; WHITESON, S. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In: **Conference on Neural Information Processing Systems (NIPS)**. 2016.

FREIRE, V.; COSTA, A. H. R. Comparative Analysis of Abstract Policies to Transfer Learning in Robotics Navigation. In: **AAAI Workshop on Knowledge, Skill, and Behavior Transfer in Autonomous Robots**. 2015. p. 9–15.

GARANT, D.; SILVA, B. C. da; LESSER, V.; ZHANG, C. Accelerating Multi-agent Reinforcement Learning with Dynamic Co-learning. **Technical Report**, 2015. Available in: <http://mas.cs.umass.edu/paper/532>.

GLATT, R.; SILVA, F. L. D.; COSTA, A. H. R. Towards Knowledge Transfer in Deep Reinforcement Learning. In: **Brazilian Conference on Intelligent Systems (BRACIS)**. 2016. p. 91–96.

GRIFFITH, S.; SUBRAMANIAN, K.; SCHOLZ, J.; ISBELL, C. L.; THOMAZ, A. L. Policy Shaping: Integrating Human Feedback with Reinforcement Learning. In: **Advances in Neural Information Processing Systems (NIPS)**. 2013. p. 2625–2633.

GUPTA, A.; DEVIN, C.; LIU, Y.; ABBEEL, P.; LEVINE, S. Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning. In: **Proceedings of the 5th International Conference on Learning Representations (ICLR)**. 2017.

GUPTA, J. K.; EGOROV, M.; KOCHENDERFER, M. Cooperative Multi-agent Control Using Deep Reinforcement Learning. In: **AAMAS Adaptive Learning Agents (ALA) Workshop**. 2017. ISBN 978-3-319-71682-4.

HANNA, J.; STONE, P. Grounded Action Transformation for Robot Learning in Simulation. In: **Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)**. 2017. p. 3834–3840.

HAUSKNECHT, M.; MUPPARAJU, P.; SUBRAMANIAN, S.; KALYANAKRISHNAN, S.; STONE, P. Half Field Offense: An Environment for Multiagent Learning and Ad Hoc Teamwork. In: **AAMAS Adaptive Learning Agents (ALA) Workshop**. 2016.

HAUWERE, Y.-M. D.; VRANCX, P.; NOWÉ, A. Learning Multi-Agent State Space Representations. In: **Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2010. p. 715–722.

HERNANDEZ-LEAL, P.; KAISERS, M. Towards a Fast Detection of Opponents in Repeated Stochastic Games. In: **Proceedings of the 1st Workshop on Transfer in Reinforcement Learning (TiRL)**. 2017.

HERNANDEZ-LEAL, P.; KAISERS, M.; BAARSLAG, T.; COTE, E. M. de. A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity. **arXiv:1707.09183**, 2017. Available in: <http://arxiv.org/abs/1707.09183>.

HESSEL, M.; MODAYIL, J.; HASSELT, H. V.; SCHAUL, T.; OSTROVSKI, G.; DABNEY, W.; HORGAN, D.; PIOT, B.; AZAR, M.; SILVER, D. Rainbow: Combining Improvements in Deep Reinforcement Learning. In: **Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)**. 2018. p. 3215–3222.

HU, J.; WELLMAN, M. P. Nash Q-learning for General-Sum Stochastic Games. **Journal of Machine Learning Research (JMLR)**, JMLR. org, v. 4, p. 1039–1069, 2003.

HU, Y.; GAO, Y.; AN, B. Accelerating Multiagent Reinforcement Learning by Equilibrium Transfer. **IEEE Transactions on Cybernetics**, IEEE, v. 45, n. 7, p. 1289–1302, 2015.

HU, Y.; GAO, Y.; AN, B. Learning in Multi-agent Systems with Sparse Interactions by Knowledge Transfer and Game Abstraction. In: **Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2015. p. 753–761.

ISELE, D.; ROSTAMI, M.; EATON, E. Using Task Features for Zero-Shot Knowledge Transfer in Lifelong Learning. In: **Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)**. 2016. p. 1620–1626.

JUDAH, K.; FERN, A. P.; DIETTERICH, T. G.; TADEPALLI, P. Active Imitation Learning: Formal and Practical Reductions to I.I.D. Learning. **Journal of Machine Learning Research (JMLR)**, v. 15, n. 1, p. 3925–3963, 2014.

JUDAH, K.; ROY, S.; FERN, A.; DIETTERICH, T. G. Reinforcement Learning Via Practice and Critique Advice. In: **Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)**. 2010. p. 481–486.

KELLY, S.; HEYWOOD, M. I. Knowledge Transfer from Keepaway Soccer to Half-Field Offense through Program Symbiosis: Building Simple Programs for a Complex Task. In: ACM. **Proceedings of the 17th Conference on Genetic and Evolutionary Computation (GECCO)**. 2015. p. 1143–1150.

KERSTING, K.; OTTERLO, M. van; RAEDT, L. D. Bellman Goes Relational. In: **Proceedings of the 21st International Conference on Machine Learning (ICML)**. 2004. p. 465–472. Available in: <http://doi.acm.org/10.1145/1015330.1015401>.

KITANO, H.; ASADA, M.; KUNIYOSHI, Y.; NODA, I.; OSAWA, E.; MATSUBARA, H. RoboCup: A Challenge Problem for AI. **AI magazine**, v. 18, n. 1, p. 73, 1997.

KNOX, W. B.; STONE, P. Interactively Shaping Agents via Human Reinforcement: The TAMER Framework. In: **Proceedings of the 5th International Conference on Knowledge Capture**. 2009. p. 9–16.

KOGA, M. L.; SILVA, V. F. da; COSTA, A. H. R. Stochastic Abstract Policies: Generalizing Knowledge to Improve Reinforcement Learning. **IEEE Transactions on Cybernetics**, v. 45, n. 1, p. 77–88, 2015.

KOGA, M. L.; SILVA, V. F. da; COZMAN, F. G.; COSTA, A. H. R. Speeding-up Reinforcement Learning Through Abstraction and Transfer Learning. In: **Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2013. p. 119–126. ISBN 978-1-4503-1993-5.

KOLTER, J. Z.; ABBEEL, P.; NG, A. Y. Hierarchical Apprenticeship Learning with Application to Quadruped Locomotion. In: **Advances in Neural Information Processing Systems (NIPS)**. 2008. p. 769–776.

KONO, H.; KAMIMURA, A.; TOMITA, K.; MURATA, Y.; SUZUKI, T. Transfer Learning Method Using Ontology for Heterogeneous Multi-agent Reinforcement Learning. **International Journal of Advanced Computer Science and Applications (IJACSA)**, v. 5, n. 10, p. 156–164, 2014.

KRENING, S.; HARRISON, B.; FEIGH, K. M.; ISBELL, C. L.; RIEDL, M.; THOMAZ, A. Learning from Explanations Using Sentiment and Advice in RL. **IEEE Transactions on Cognitive and Developmental Systems**, v. 9, n. 1, p. 44–55, March 2017. ISSN 2379-8920.

LAUER, M.; RIEDMILLER, M. An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems. In: **Proceedings of the 17th International Conference on Machine Learning (ICML)**. 2000. p. 535–542.

LAZARIC, A. Transfer in Reinforcement Learning: A Framework and a Survey. In: ____. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. p. 143–173. ISBN 978-3-642-27645-3.

LE, H. M.; YUE, Y.; CARR, P. Coordinated Multi-Agent Imitation Learning. In: **Proceedings of the 34th International Conference on Machine Learning (ICML)**. 2017. p. 1995–2003.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep Learning. **Nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LEVINE, S.; FINN, C.; DARRELL, T.; ABBEEL, P. End-to-end Training of Deep Visuomotor Policies. **The Journal of Machine Learning Research**, JMLR. org, v. 17, n. 1, p. 1334–1373, 2016.

LIN, X.; BELING, P. A.; COGILL, R. Multiagent Inverse Reinforcement Learning for Two-Person Zero-Sum Games. **IEEE Transactions on Games**, v. 10, n. 1, p. 56–68, 2018.

LITTMAN, M. L. Markov Games as a Framework for Multi-Agent Reinforcement Learning. In: **Proceedings of the 11th International Conference on Machine Learning (ICML)**. 1994. p. 157–163.

LITTMAN, M. L. Reinforcement Learning Improves Behaviour from Evaluative Feedback. **Nature**, v. 521, n. 7553, p. 445–451, 2015. ISSN 0028-0836.

LOPES, M.; MELO, F.; MONTESANO, L. Active Learning for Reward Estimation in Inverse Reinforcement Learning. In: SPRINGER. **Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)**. 2009. p. 31–46.

MACGLASHAN, J. **Brown-UMBC Reinforcement Learning and Planning (BURLAP).** 2015. Available in: <http://burlap.cs.brown.edu/index.html>.

MACGLASHAN, J.; HO, M. K.; LOFTIN, R.; PENG, B.; WANG, G.; ROBERTS, D. L.; TAYLOR, M. E.; LITTMAN, M. L. Interactive Learning from Policy-Dependent Human Feedback. In: **Proceedings of the 34th International Conference on Machine Learning (ICML)**. 2017. p. 2285–2294.

MACLIN, R.; SHAVLIK, J. W.; KAELBLING, P. Creating Advice-Taking Reinforcement Learners. In: **Machine Learning**. 1996. v. 22, n. 1-3, p. 251–281.

MADDEN, M. G.; HOWLEY, T. Transfer of Experience Between Reinforcement Learning Environments with Progressive Difficulty. **Artificial Intelligence Review**, v. 21, n. 3, p. 375–398, Jun 2004. ISSN 1573-7462.

MANDEL, T.; LIU, Y.-E.; BRUNSKILL, E.; POPOVIC, Z. Where to Add Actions in Human-in-the-Loop Reinforcement Learning. In: **Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)**. 2017. p. 2322–2328.

MATIISEN, T.; OLIVER, A.; COHEN, T.; SCHULMAN, J. Teacher-Student Curriculum Learning. **arXiv:1707.00183**, 2017.

MELO, F. S.; VELOSO, M. Decentralized MDPs with Sparse Interactions. **Artificial Intelligence**, v. 175, n. 11, p. 1757 – 1789, 2011. ISSN 0004-3702.

MNIH, V.; BADIA, A. P.; MIRZA, M.; GRAVES, A.; LILLICRAP, T.; HARLEY, T.; SILVER, D.; KAVUKCUOGLU, K. Asynchronous Methods for Deep Reinforcement Learning. In: **Proceedings of the 33rd International Conference on Machine Learning (ICML)**. 2016. p. 1928–1937.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G.; PETERSEN, S.; BEATTIE, C.; SADIK, A.; ANTONOGLOU, I.; KING, H.; KUMARAN, D.; WIERSTRA, D.; LEGG, S.; HASSABIS, D. Human-level Control through Deep Reinforcement Learning. **Nature**, v. 518, n. 7540, p. 529–533, 2015. Available in: <http://dx.doi.org/10.1038/nature14236>.

NARVEKAR, S.; SINAPOV, J.; LEONETTI, M.; STONE, P. Source Task Creation for Curriculum Learning. In: **Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2016. p. 566–574.

NARVEKAR, S.; SINAPOV, J.; STONE, P. Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning. In: **Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)**. 2017. p. 2536–2542.

NATARAJAN, S.; KUNAPULI, G.; JUDAH, K.; TADEPALLI, P.; KERSTING, K.; SHAVLIK, J. Multi-Agent Inverse Reinforcement Learning. In: IEEE. **Proceedings of the 9th International Conference on Machine Learning and Applications (ICMLA)**. 2010. p. 395–400.

NG, A. Y.; COATES, A.; DIEL, M.; GANAPATHI, V.; SCHULTE, J.; TSE, B.; BERGER, E.; LIANG, E. Autonomous Inverted Helicopter Flight via Reinforcement Learning. In: **Experimental Robotics IX**. : Springer, 2006. p. 363–372.

NG, A. Y.; HARADA, D.; RUSSELL, S. Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping. In: **Proceedings of the 16th International Conference on Machine Learning (ICML)**. 1999. p. 278–287.

NUNES, L.; OLIVEIRA, E. On learning by exchanging advice. **Journal of Artificial Intelligence and the Simulation of Behaviour**, v. 1, n. 3, p. 241–257, July 2003.

OMIDSHAFIEI, S.; KIM, D.; LIU, M.; TESAURO, G.; RIEMER, M.; AMATO, C.; CAMPBELL, M.; HOW, J. P. Learning to Teach in Cooperative Multiagent Reinforcement Learning. In: **Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)**. 2019.

OMIDSHAFIEI, S.; PAZIS, J.; AMATO, C.; HOW, J. P.; VIAN, J. Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability. In: **Proceedings of the 34th International Conference on Machine Learning (ICML)**. 2017. p. 2681–2690.

OSBAND, I.; BLUNDELL, C.; PRITZEL, A.; ROY, B. V. Deep Exploration via Bootstrapped DQN. In: **Advances in Neural Information Processing Systems (NIPS)**. 2016. p. 4026–4034.

PAN, S. J.; YANG, Q. A Survey on Transfer Learning. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 22, n. 10, p. 1345–1359, 2010.

PANAIT, L.; LUKE, S. Cooperative Multi-Agent Learning: The State of the Art. **Autonomous Agents and Multiagent Systems**, v. 11, n. 3, p. 387–434, 2005.

PENG, B.; MACGLASHAN, J.; LOFTIN, R.; LITTMAN, M. L.; ROBERTS, D. L.; TAYLOR, M. E. A Need for Speed: Adapting Agent Action Speed to Improve Task Learning from Non-Expert Humans. In: **Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2016. p. 957–965.

PENG, B.; MACGLASHAN, J.; LOFTIN, R.; LITTMAN, M. L.; ROBERTS, D. L.; TAYLOR, M. E. An Empirical Study of Non-expert Curriculum Design for Machine Learners. In: **Proceedings of the IJCAI Interactive Machine Learning Workshop**. 2016.

PERICO, D. H.; BIANCHI, R. A. C. Use of Heuristics from Demonstrations to Speed Up Reinforcement Learning. In: **Proceedings of the 12th Brazilian Symposium on Intelligent Automation (SBAI)**. 2013. Available in: <http://www.sbai2013.ufc.br/pdfs/4908.pdf>.

PRICE, B.; BOUTILIER, C. Accelerating Reinforcement Learning through Implicit Imitation. **Journal of Artificial Intelligence Research (JAIR)**, v. 19, p. 569–629, 2003. Available in: <http://arxiv.org/abs/1709.04083>.

PROPER, S.; TADEPALLI, P. Multiagent Transfer Learning via Assignment-Based Decomposition. In: **Proceedings of the 8th International Conference on Machine Learning and Applications (ICMLA)**. 2009. p. 345–350.

PUTERMAN, M. L. **Markov Decision Processes : Discrete Stochastic Dynamic Programming**. Hoboken (N. J.): J. Wiley & Sons, 2005. ISBN 0-471-72782-2.

RAMACHANDRAN, D.; AMIR, E. Bayesian Inverse Reinforcement Learning. In: **Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI)**. San Francisco, CA, USA: , 2007. p. 2586–2591.

RAMAKRISHNAN, R.; NARASIMHAN, K.; SHAH, J. Interpretable Transfer for Reinforcement Learning based on Object Similarities. In: **Proceedings of the IJCAI Interactive Machine Learning Workshop**. 2016.

REDDY, T. S.; GOPIKRISHNA, V.; ZARUBA, G.; HUBER, M. Inverse Reinforcement Learning for Decentralized Non-Cooperative Multiagent Systems. In: **Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC)**. 2012. p. 1930–1935.

ROSENFELD, A.; TAYLOR, M. E.; KRAUS, S. Leveraging Human Knowledge in Tabular Reinforcement Learning: A Study of Human Subjects. In: **Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)**. 2017. p. 3823–3830.

SAKATO, T.; OZEKI, M.; OKA, N. Learning through Imitation and Reinforcement Learning: Toward the Acquisition of Painting Motions. In: **Proceedings of the 3rd International Conference on Advanced Applied Informatics (IIAI)**. 2014. p. 873–880.

SCHAAL, S. Learning from Demonstration. In: **Advances in Neural Information Processing Systems (NIPS)**. 1997. p. 1040–1046.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural Networks**, Elsevier, v. 61, p. 85–117, 2015.

SHERSTOV, A. A.; STONE, P. Improving Action Selection in MDP's via Knowledge Transfer. In: AAAI PRESS. **Proceedings of the 20th AAAI Conference on Artificial Intelligence (AAAI)**. 2005. p. 1024–1029.

SHON, A. P.; VERMA, D.; RAO, R. P. N. Active Imitation Learning. In: **Proceedings of the 21st AAAI Conference on Artificial Intelligence (AAAI)**. 2007. p. 756–762.

SILVA, F. L. D. Integrating Agent Advice and Previous Task Solutions in Multiagent Reinforcement Learning. In: **Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2019.

SILVA, F. L. D.; COSTA, A. H. R. Transfer Learning for Multiagent Reinforcement Learning Systems. In: **Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)**. 2016. p. 3982–3983.

SILVA, F. L. D.; COSTA, A. H. R. Accelerating Multiagent Reinforcement Learning through Transfer Learning. In: **Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)**. 2017. p. 5034–5035.

SILVA, F. L. D.; COSTA, A. H. R. Automatic Object-Oriented Curriculum Generation for Reinforcement Learning. In: **Proceedings of the 1st Workshop on Scaling Up Reinforcement Learning (SURL)**. 2017.

SILVA, F. L. D.; COSTA, A. H. R. Towards Zero-Shot Autonomous Inter-Task Mapping through Object-Oriented Task Description. In: **Proceedings of the 1st Workshop on Transfer in Reinforcement Learning (TiRL)**. 2017.

SILVA, F. L. D.; COSTA, A. H. R. Object-Oriented Curriculum Generation for Reinforcement Learning. In: **Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2018. p. 1026–1034.

SILVA, F. L. D.; COSTA, A. H. R. A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems. **Journal of Artificial Intelligence Research (JAIR)**, v. 69, p. 645–703, 2019.

SILVA, F. L. D.; GLATT, R.; COSTA, A. H. R. Object-Oriented Reinforcement Learning in Cooperative Multiagent Domains. In: **Proceedings of the 5th Brazilian Conference on Intelligent Systems (BRACIS)**. 2016. p. 19–24.

SILVA, F. L. D.; GLATT, R.; COSTA, A. H. R. An advising framework for multiagent reinforcement learning systems. In: **AAAI-17**. 2017. p. (Accepted).

SILVA, F. L. D.; GLATT, R.; COSTA, A. H. R. Simultaneously Learning and Advising in Multiagent Reinforcement Learning. In: **Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2017. p. 1100–1108.

SILVA, F. L. D.; GLATT, R.; COSTA, A. H. R. MOO-MDP: An Object-Oriented Representation for Cooperative Multiagent Reinforcement Learning. **IEEE Transactions on Cybernetics**, v. 49, n. 2, p. 567–579, 2019. ISSN 2168-2267.

SILVA, F. L. D.; HERNANDEZ-LEAL, P.; KARTAL, B.; TAYLOR, M. E. Uncertainty-Aware Action Advising for Deep Reinforcement Learning Agents. In: **Deep Reinforcement Learning Workshop at NeurIPS**. 2019.

SILVA, F. L. D.; TAYLOR, M. E.; COSTA, A. H. R. Autonomously Reusing Knowledge in Multiagent Reinforcement Learning. In: **Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)**. 2018. p. 5487–5493.

SILVA, F. L. D.; WARNELL, G.; COSTA, A. H. R.; STONE, P. Agents Teaching Agents: Inter-agent Transfer Learning. **Autonomous Agents and Multi-Agent Systems**, in press, 2019.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. van den; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M.; DIELEMAN, S.; GREWE, D.; NHAM, J.; KALCHBRENNER, N.; SUTSKEVER, I.; LILLICRAP, T.; LEACH, M.; KAVUKCUOGLU, K.; GRAEPEL, T.; HASSABIS, D. Mastering the Game of Go with Deep Neural

Networks and Tree Search. **Nature**, v. 529, p. 484–503, 2016. Available in: <http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html>.

SINAPOV, J.; NARVEKAR, S.; LEONETTI, M.; STONE, P. Learning Inter-Task Transferability in the Absence of Target Task Samples. In: **Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2015. p. 725–733.

SODOMKA, E.; HILLIARD, E.; LITTMAN, M. L.; GREENWALD, A. Coco-Q: Learning in Stochastic Games with Side Payments. In: **Proceedings of the 30th International Conference on Machine Learning (ICML)**. 2013. v. 28, p. 1471–1479.

STONE, P.; KAMINKA, G. A.; KRAUS, S.; ROSENSCHEIN, J. S. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In: **Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI)**. 2010. p. 1504–1509.

STONE, P.; SUTTON, R. S.; KUHLMANN, G. Reinforcement Learning for RoboCup-Soccer Keepaway. **Adaptive Behavior**, v. 13, n. 3, p. 165–188, 2005.

STONE, P.; VELOSO, M. Multiagent Systems: A Survey from a Machine Learning Perspective. **Autonomous Robots**, v. 8, n. 3, p. 345–383, July 2000.

SUAY, H. B.; BRYS, T.; TAYLOR, M. E.; CHERNOVA, S. Learning from Demonstration for Shaping Through Inverse Reinforcement Learning. In: **Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2016. p. 429–437. ISBN 978-1-4503-4239-1.

SUBRAMANIAN, K.; JR, C. L. I.; THOMAZ, A. L. Exploration from Demonstration for Interactive Reinforcement Learning. In: **Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2016. p. 447–456.

SUKHBAATAR, S.; KOSTRIKOV, I.; SZLAM, A.; FERGUS, R. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. In: **Proceedings of the 6th International Conference on Learning Representations (ICLR)**. 2018.

SUKHBAATAR, S.; SZLAM, A.; FERGUS, R. Learning Multiagent Communication with Backpropagation. In: **Conference on Neural Information Processing Systems (NIPS)**. 2016.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. 1st. ed. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262193981.

SVETLIK, M.; LEONETTI, M.; SINAPOV, J.; SHAH, R.; WALKER, N.; STONE, P. Automatic Curriculum Graph Generation for Reinforcement Learning Agents. In: **Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)**. San Francisco, CA: , 2017. p. 2590–2596.

TAMASSIA, M.; ZAMBETTA, F.; RAFFE, W.; MUELLER, F.; LI, X. Learning Options from Demonstrations: A Pac-Man Case Study. **IEEE Transactions on Computational Intelligence and AI in Games**, v. 10, n. 1, p. 91–96, 2017. ISSN 1943-068X.

TAN, M. Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents. In: **Proceedings of the 10th International Conference on Machine Learning (ICML)**. 1993. p. 330–337.

TAYLOR, A.; DUSPARIC, I.; GALVAN-LOPEZ, E.; CLARKE, S.; CAHILL, V. Accelerating Learning in Multi-Objective Systems through Transfer Learning. In: **International Joint Conference on Neural Networks (IJCNN)**. 2014. p. 2298–2305.

TAYLOR, M. E.; CARBONI, N.; FACHANTIDIS, A.; VLAHAVAS, I. P.; TORREY, L. Reinforcement Learning Agents Providing Advice in Complex Video Games. **Connection Science**, v. 26, n. 1, p. 45–63, 2014.

TAYLOR, M. E.; STONE, P. Transfer Learning for Reinforcement Learning Domains: A Survey. **Journal of Machine Learning Research (JMLR)**, v. 10, p. 1633–1685, 2009.

TAYLOR, M. E.; STONE, P.; LIU, Y. Transfer Learning via Inter-Task Mappings for Temporal Difference Learning. **Journal of Machine Learning Research (JMLR)**, v. 8, n. 1, p. 2125–2167, 2007.

TAYLOR, M. E.; WHITESON, S.; STONE, P. Transfer via Inter-Task Mappings in Policy Search Reinforcement Learning. In: **Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2007.

TESAURO, G. Temporal Difference Learning and TD-Gammon. **Commun. ACM**, ACM, New York, NY, USA, v. 38, n. 3, p. 58–68, Mar. 1995. ISSN 0001-0782.

THRUN, S.; MITCHELL, T. M. Lifelong Robot Learning. **Robotics and Autonomous Systems**, Elsevier, v. 15, n. 1-2, p. 25–46, 1995.

TORREY, L.; TAYLOR, M. E. Teaching on a Budget: Agents Advising Agents in Reinforcement Learning. In: **Proceedings of 12th the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**. 2013. p. 1053–1060.

VAMPLEW, P.; DAZELEY, R.; BERRY, A.; ISSABEKOV, R.; DEKKER, E. Empirical Evaluation Methods for Multiobjective Reinforcement Learning Algorithms. **Machine Learning**, v. 84, n. 1, p. 51–80, Jul 2011. ISSN 1573-0565.

VRANCX, P.; HAUWERE, Y. D.; NOWÉ, A. Transfer Learning for Multi-agent Coordination. In: **Proceedings of the 3rd International Conference on Agents and Artificial Intelligence (ICAART)**. 2011. p. 263–272.

WALSH, T. J.; HEWLETT, D. K.; MORRISON, C. T. Blending Autonomous Exploration and Apprenticeship Learning. In: **Advances in Neural Information Processing Systems (NIPS)**. Curran Associates, Inc., 2011. p. 2258–2266. Available in: <http://papers.nips.cc/paper/4240-blending-autonomous-exploration-and-apprenticeship-learning.pdf>.

WANG, G.-f.; FANG, Z.; LI, P.; LI, B. Transferring Knowledge from Human-Demonstration Trajectories to Reinforcement Learning. **Transactions of the Institute of Measurement and Control**, v. 40, n. 1, p. 94–101, 2016.

WANG, Z.; TAYLOR, M. E. Improving Reinforcement Learning with Confidence-Based Demonstrations. In: **Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)**. 2017. p. 3027–3033.

WATKINS, C. J.; DAYAN, P. Q-Learning. **Machine Learning**, Springer Netherlands, v. 8, n. 3, p. 279–292, 1992.

WIEWIORA, E.; COTTRELL, G. W.; ELKAN, C. Principled Methods for Advising Reinforcement Learning Agents. In: **Proceedings of the 20th International Conference on Machine Learning (ICML)**. 2003. p. 792–799.

WOOLDRIDGE, M. J. **An Introduction to MultiAgent Systems (2. ed.)**. : Wiley, 2009. ISBN 978-0-470-51946-2.

XIONG, Y.; CHEN, H.; ZHAO, M.; AN, B. HogRider: Champion Agent of Microsoft Malmo Collaborative AI Challenge. In: **Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)**. 2018. p. 4767–4774.

ZHAN, Y.; BOU-AMMAR, H.; TAYLOR, M. E. Theoretically-Grounded Policy Advice from Multiple Teachers in Reinforcement Learning Settings with Applications to Negative Transfer. In: **Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)**. 2016. p. 2315–2321.

ZHIFEI, S.; JOO, E. M. A Survey of Inverse Reinforcement Learning Techniques. **International Journal of Intelligent Computing and Cybernetics**, v. 5, n. 3, p. 293–311, 2012. Available in: <https://doi.org/10.1108/17563781211255862>.

ZHOU, L.; YANG, P.; CHEN, C.; GAO, Y. Multiagent Reinforcement Learning With Sparse Interactions by Negotiation and Knowledge Transfer. **IEEE Transactions on Cybernetics**, v. 47, n. 5, p. 1238–1250, 2016. ISSN 2168-2267.

Appendix A

List of Publications

Portions of the content described in this thesis have been previously published in Conferences and Journals. We list all relevant papers and articles below.

- **Silva, F. L.**; & Costa, A. H. R. (2019). *A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems*. **Journal of Artificial Intelligence Research (JAIR)**, 64, 645-703.

- **Silva, F. L.**; Glatt, R.; & Costa, A. H. R. (2019). *MOO-MDP: An Object-Oriented Representation for Cooperative Multiagent Reinforcement Learning*. **IEEE Transactions on Cybernetics**, 49(2), 567-579.

- **Silva, F. L.**; Warnell, G.; Costa, A. H. R.; & Stone, P. (2019). *Agents Teaching Agents: Inter-agent Transfer Learning*. **Autonomous Agents and Multi-Agent Systems**, submitted.

- **Silva, F. L.**; Taylor, M. E.; & Costa, A. H. R. (2018). *Autonomously Reusing Knowledge in Multiagent Reinforcement Learning*. In **International Joint Conference on Artificial Intelligence (IJCAI)**, 5487-5493.

- **Silva, F. L.**; & Costa, A. H. R. (2018). *Object-Oriented Curriculum Generation for Reinforcement Learning*. In **International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**, 1026-1034.

- **Silva, F. L.**; Glatt, R.; & Costa, A. H. R. (2017). *Simultaneously Learning and Advising in Multiagent Reinforcement Learning*. In **International Conference on Autonomous Agents and Multiagent Systems (AAMAS)**, 1100-1108.

- **Silva, F. L.**; Hernandez-Leal, P.; & Kartal, B.; Taylor, M. E. (2019). *Uncertainty-Aware Action Advising for Deep Reinforcement Learning Agents*. In **Deep Reinforcement Learning Workshop at NeurIPS**.

- **Silva, F. L.**; Costa, A. H. R.; & Stone, P. (2019). *Building Self-Play Curricula Online by Playing with Expert Agents*. In **Brazilian Conference on Intelligent Systems (BRACIS)**, accepted.

- **Silva, F. L.**; Glatt, R.; & Costa, A. H. R. (2016). *Object-Oriented Reinforcement Learning in Cooperative Multiagent Domains*. In **Brazilian Conference on Intelligent Systems (BRACIS)**, 19-24.

- **Silva, F. L.**; Costa, A. H. R.; & Stone, P. (2019). *Distributional Reinforcement Learning Applied to Robot Soccer Simulation*. In **Adaptive Learning Agents (ALA) Workshop at AAMAS**.

- **Silva, F. L.**; & Costa, A. H. R. (2017). *Automatic Object-Oriented Curriculum Generation for Reinforcement Learning*. In **Workshop on Scaling Up Reinforcement Learning (SURL)**.

- **Silva, F. L.**; & Costa, A. H. R. (2017). *Towards Zero-Shot Autonomous Inter-Task Mapping through Object-Oriented Task Description*. In **Workshop on Transfer in Reinforcement Learning (TiRL)**.

- **Silva, F. L.** (2019). *Integrating Agent Advice and Previous Task Solutions in Multiagent Reinforcement Learning*. In **International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Doctoral Consortium**.

- **Silva, F. L.**; & Costa, A. H. R. (2017). *Accelerating Multiagent Reinforcement Learning through Transfer Learning*. In **AAAI Conference on Artificial Intelligence, Doctoral Consortium**.

- **Silva, F. L.**; Glatt, R.; & Costa, A. H. R. (2017). *An Advising Framework for Multiagent Reinforcement Learning Systems*. In **AAAI Conference on Artificial Intelligence, Student Paper**.

- **Silva, F. L.**; & Costa, A. H. R. (2016). *Transfer Learning for Multiagent Reinforcement Learning Systems*. In **International Joint Conference on Artificial Intelligence (IJCAI), Doctoral Consortium**.

Appendix B

Proof for Equation (5.2)

Proposition 1. *Equation (5.2) holds for every step k, agent $z \in Ag$, state $s \in S$, and action $a^z \in A^z$, in any MOO-MDP where the following assumptions hold:*

1. *The concrete state transition and reward functions are deterministic (i.e., for a given state s_k and joint action $\boldsymbol{u_k}$ only one next state s_{k+1} and reward r_k can be achieved).*
2. *For all $s \in S, \boldsymbol{u} \in U$ and $a^z \in A^z$: $Q_0(s, \boldsymbol{u}) = Q_0^z(\kappa(s, z), a^z) = 0$, and $r(s, \boldsymbol{u}) \geq 0$.*
3. *The MOO-MDP is cooperative (i.e., all agents receive the same reward r_k at every step k).*
4. *For all $s \in S$, $z \in G$, and $\boldsymbol{u} \in U$, $\kappa(s, z)$ returns only one abstract state $\tilde{s}_k^z = \kappa(s, z)$. Also, the same reward r_k and next state $\tilde{s}_{k+1}^z$ are observed when applying $\boldsymbol{u}$ in any concrete state covered by $\tilde{s}_k^z$.*

Proof. For all agents $z \in G$:

- $k = 0$: Equation (5.2) is ensured by Assumption 2.
- $k \rightsquigarrow k + 1$: Assumptions 1 and 3 ensure that the same experience $\langle s_k, \boldsymbol{u_k}, s_{k+1}, r_k \rangle$ is valid for all agents in k, which together with Assumption 4 guarantees that each agent always observes the same $\tilde{s}_k^z$ whenever s_k is visited. Thus, the following Q-value update is performed either in distributed or in joint control, respectively:

$$Q_{k+1}^z(\tilde{s}_k^z, a_k^z) \leftarrow \max\{Q_k^z(\tilde{s}_k^z, a_k^z), r_k + \gamma \max_{a^z \in A_z} Q_k^z(\tilde{s}_{k+1}^z, a^z)\},$$
$$Q_{k+1}(s_k, \boldsymbol{u}_k) \leftarrow \max\{Q_k(s_k, \boldsymbol{u}_k), r_k + \gamma \max_{\boldsymbol{u} \in U} Q_k(s_{k+1}, \boldsymbol{u})\}.$$

For any experience, this update can lead to two possibilities:

1. $Q_k^z(\tilde{s}_k^z, a_k^z) < r_k + \gamma \max_{a^z \in A_z} Q_k^z(\tilde{s}_{k+1}^z, a^z)$: Because Equation (5.2) holds for k, both $Q_k^z(\tilde{s}_k^z, a_k^z)$ and $Q_k(s, \boldsymbol{u_k})$ are updated, thus after the update: $Q_{k+1}^z(\tilde{s}_k^z, a_k^z) \geq Q_{k+1}(s_k, \boldsymbol{u_k})$.
2. *Otherwise*: No Q-value is updated on the distributed Q-table. As Equation (5.2) holds for k: $Q_k(s_k, \boldsymbol{u_k}) \leq Q_k^z(\tilde{s}_k^z, a_k^z)$ and $\max_{\boldsymbol{u} \in U, s \in \tilde{s}_{k+1}^z} Q_k(s, \boldsymbol{u}) \leq \max_{a^z \in A_z} Q_k^z(\tilde{s}_{k+1}^z, a^z)$. This means that, after the update in k, the following relation is valid: $Q_{k+1}(s_k, \boldsymbol{u}_k) \leq Q_{k+1}^z(\tilde{s}_k^z, a_k^z)$.

Since in both situations $Q_{k+1}(s_k, \boldsymbol{u}_k)$ and $Q_{k+1}^z(\tilde{s}_k^z, a_k^z)$ are the only Q-table entries that may be updated and the latter is always greater or equal than the former, Equation (5.2) also holds for $k + 1$.

Assumptions 1, 3, and 4 also ensure that at convergence time, all trajectories starting from a given concrete state s are already known, resulting in the following equation for any $z \in G$, $s \in S$, and $a^z \in A^z$:

$$Q^{z*}(\kappa(s, z), a^z) = \max_{\boldsymbol{u} \in U, \boldsymbol{u}^z = a^z} Q^*(s, u), \tag{B.1}$$

where Q^{z*} and Q^* are, respectively, the distributed Q-table of agent z and the joint Q-table at convergence time.

□

Appendix C

Proof for Learning a Greedy Joint Policy with DOO-Q

Proposition 2. *Let π^z be a decentralized policy learned by agent z on a cooperative MOO-MDP using Equation (5.3). Assume that Equation (5.2) holds and Equation (5.1) is used for Q-value updates. Let $\tilde{s}_k^z = \kappa(s_k, z)$, then for every state $s \in S$, π^z is greedy with respect to the corresponding joint Q-table at convergence time, i.e.*

$$\forall s \in S : [\pi^1(\tilde{s}^1) \ \dots \ \pi^m(\tilde{s}^m)]^T = argmax_{\boldsymbol{u} \in U} Q^*(s, \boldsymbol{u}), \tag{C.1}$$

where Q^* is the joint Q-table at convergence time.

Proof. For all agents $z \in G$, let π_0^z be arbitrarily initialized. Because Equation (5.1) is used for Q-value updates, Q_k^z is a monotonically increasing function; that is, $\forall s \in S, a^z \in A^z : Q_k^z(s, a^z) \leq Q_{k+1}^z(s, a^z)$. However, according to Equation (5.3), the policy is only updated in step k, when there exists only one action for which the Q-value related to the current state was modified:

$$\exists! a^z \in A_z : Q_k^z(\tilde{s}_k^z, a^z) < Q_{k+1}^z(\tilde{s}_k^z, a^z).$$

In this case, we know that: $\pi_{k+1}^z(\tilde{s}_k^z) \leftarrow a_k^z$. As we are dealing with cooperative MOO-MDPs, this holds for all agents in k, corresponding to a joint policy update as follows

$$\pi_{k+1}(s_k) = \begin{bmatrix} \pi_{k+1}^1(\tilde{s}_k^1) \\ \vdots \\ \pi_{k+1}^m(\tilde{s}_k^m) \end{bmatrix} = \begin{bmatrix} \underset{a^1 \in A^1}{arg\ max} Q_{k+1}^1(\tilde{s}_k^1, a^1) \\ \vdots \\ \underset{a^m \in A^m}{arg\ max} Q_{k+1}^m(\tilde{s}_k^m, a^m) \end{bmatrix} \tag{C.2}$$

which means that, at convergence time, the resulting joint policy corresponds to

$$\forall s \in S : \pi^*(s) = \begin{bmatrix} \underset{a^1 \in A^1}{arg\ max} Q^{1*}(\tilde{s}^1, a^1) \\ \vdots \\ \underset{a^m \in A^m}{arg\ max} Q^{m*}(\tilde{s}^m, a^m) \end{bmatrix}. \tag{C.3}$$

Combining Equations (B.1) and (C.3) results in

$$\forall s \in S : \pi^*(s) = \begin{bmatrix} \underset{a^1 \in A^1}{arg\ max} \underset{\boldsymbol{u} \in U, \boldsymbol{u^1} = a^1}{max} Q^*(s, u) \\ \vdots \\ \underset{a^m \in A^m}{arg\ max} \underset{\boldsymbol{u} \in U, \boldsymbol{u^m} = a^m}{max} Q^*(s, u) \end{bmatrix}. \tag{C.4}$$

Due to the update rule in Equation (5.3) and the cooperative nature of the MOO-MDP, we can say that agents coordinate by breaking ties in $\underset{a^z \in A^i}{arg\ max} Q^i_{k+1}(\tilde{s}^i, a^z)$ according to the order in which experiences occurred, which means that agents coordinate even when multiple optimal joint policies exist. Thus Equation (C.4) is equivalent to

$$\forall s \in S : \pi^*(s) = \underset{\boldsymbol{u} \in U}{arg\ max}\ Q^*(s, u). \tag{C.5}$$

Hence, a joint policy implied by decentralized policies updated as in Equation (5.3) eventually converges to the optimal joint policy, provided that all states are infinitely visited and all applicable actions have a *non-zero probability* of being executed by the exploration strategy. □