UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RICARDO GRUNITZKI

A Flexible Approach for Optimal Rewards in Multi-Agent Reinforcement Learning Problems

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Profa. Dra. Ana L.C. Bazzan

Porto Alegre
August 2018

CIP — CATALOGING-IN-PUBLICATION

Grunitzki, Ricardo

A Flexible Approach for Optimal Rewards in Multi-Agent Reinforcement Learning Problems / Ricardo Grunitzki. – Porto Alegre: PPGC da UFRGS, 2018.

91 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2018. Advisor: Ana L.C. Bazzan.

1. Optimal reward problem. 2. Reward function design. 3. Multi-agent reinforcement learning. I. Bazzan, Ana L.C.. II. Título.

“Quando penso que já cheguei ao meu limite,
descubro que tenho forças para ir além.”
— AYRTON SENNA

ACKNOWLEDGMENTS

I would like to thank you all the people that helped me to make this research possible.

Firstly, I express my gratitude to my advisor Ana L. C. Bazzan for her intellectual and personal support through this entire process. Besides my advisor, a special thank you to Professor Bruno C. da Silva who, even informally, played the role of co-advisor masterfully.

I would like to extend my gratitude to the institution UFRGS and the people that are part of it.

Thank you also to the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) for founding my research.

I would like to thank you, my friends and colleagues, from MASLAB. In special, to Gabriel Ramos, Fernando dos Santos and Liza Lemos for the mutual and immeasurable support during the development of this thesis. I also would like to thank you all my colleagues from UFRGS.

I would like to thank you all my friends. In particular, my home-mates Sergio Montazolli, Pedro Heleno, Anderson Tavares and to Matthewzão and Eric whom we shared good times during my days in Porto Alegre.

Last but not the least, I would like to thank you my parents, Marco and Claudete. Without you, I would not have been able to.

ABSTRACT

Defining a reward function that, when optimized, results in rapid acquisition of an optimal policy, is one of the most challenging tasks involved in applying reinforcement learning algorithms. The behavior learned by agents is directly related to the reward function they are using. Existing work on the *Optimal Reward Problem* (ORP) propose mechanisms to design reward functions. However, their application is limited to specific sub-classes of single or multi-agent reinforcement learning problems. Moreover, these methods identify which rewards should be given in which situation, but not which aspects of the state or environment should be used when defining the reward function.

This thesis proposes an *extended* version of the optimal reward problem (EORP) that: i) can identify both features and reward signals that should compose the reward function; ii) is general enough to deal with single and multi-agent reinforcement learning problems; iii) is scalable to problems with large number of agents learning simultaneously; iv) incorporates a *learning effort* metric in the evaluation of reward functions, allowing the discovery of reward functions that result in faster learning.

The method is evaluated on gridworld and traffic assignment problems to demonstrate its efficacy in designing effective reward functions. The results obtained by the proposed approach are compared to reward functions designed by a domain specialist and to a well-known new design technique for multi-agent rewards called difference rewards. Results show that EORP can identify reward functions that outperform these two types of reward functions in the evaluated problems.

Keywords: Optimal reward problem. Reward function design. Multi-agent reinforcement learning.

Uma Abordagem Flexível para Recompensas Ótimas em Problemas de Aprendizado por Reforço Multiagente

RESUMO

Definir uma função de recompensa que, quando otimizada, resulta em uma rápida aquisição de política ótima é uma das tarefas mais desafiadoras envolvendo o uso de algoritmos de aprendizado por reforço. O comportamento aprendido pelos agentes está diretamente relacionado à função de recompensa que eles estão utilizando. Trabalhos existentes sobre o *Optimal Reward Problem* (ORP) propõem mecanismos para projetar funções de recompensa. Entretanto, a aplicação de tais métodos é limitada à algumas subclasses específicas de problemas de aprendizado por reforço mono ou multiagente. Além do mais, os métodos em questão apenas identificam "*o quanto*" que um agente deve ser recompensado em cada situação, mas não "*quais os*" aspectos do estado ou ambiente que devem ser utilizados na estrutura da função de recompensa.

Nesta tese, nós propomos melhorias no ORP tradicional, definindo uma versão estendida do optimal reward problem (EORP) que: i) pode identificar tanto as características do estado/ambiente quanto os sinais de recompensa que compõem a função de recompensa; ii) é geral o suficiente para lidar com problemas de aprendizado por reforço mono e multiagente; iii) é escalável para problemas onde existem grandes quantidades de agentes aprendendo simultaneamente; iv) incorpora uma *métrica de esforço* de aprendizagem na avaliação das funções de recompensa, permitindo a descoberta de funções de recompensa que resultam em um aprendizado mais rápido.

Para demonstrar a eficácia do nosso método em projetar funções de recompensa efetivas, nós o avaliamos em dois cenários, onde os resultados são comparados com outras duas funções de recompensa: uma definida manualmente por um especialista de domínio e uma função do tipo *difference rewards*. Os resultados mostram que a nossa abordagem consegue identificar funções de recompensa que aprendem políticas de maior performance e que resultam em menor tempo de aprendizagem.

Palavras-chave: Problema de recompensa ótima. Projeto de função de recompensa. Aprendizado por reforço multiagente.

LIST OF ABBREVIATIONS AND ACRONYMS

CMOTP	Coordinated Multi-Agent Object Transportation Problem
DR	Difference Rewards
IRL	Inverse Reinforcement Learning
MARL	Multi-Agent Reinforcement Learning
ORF	Optimal Reward Function
ORP	Optimal Reward Problem
PBRS	Potential-Based Reward Shaping
RL	Reinforcement Learning
SOTP	Single-Agent Object Transportation Problem
TAP	Traffic Assignment Problem

LIST OF SYMBOLS

R	Reward function
S	Set of states
A	Set of actions
Q	Value function or Q-value
r	Reward signal
Λ	Maximum number of episodes
D	Difference rewards function
$G(z)$	System's utility function
$G(z_i)$	System's utility function of a hypothetical system without agent i
i	Agent
I	Set of agents
F	Fitness function
h	History generated by an agent
H	Set of the histories generated by all agents in I
J	Set of reward features
R^*	Optimal reward function
f_1	Fitness goal function
f_2	Learning effort goal function
$\mathcal{R}$	Space of reward functions
$\mathcal{F}$	Multi-objective evaluation function
$s(j)$	Indicates whether the situation j is active in a given state
$w(j)$	Reward signal of a given reward feature
$P(j)$	Indicates whether feature j is used in a given reward function.
Φ	Function that composes a reward function

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Reinforcement learning (RL) deals with problems where an agent tries to learn a behavior through successive interactions with an environment (SUTTON; BARTO, 1998). The behavior of an agent is represented by a policy that maps states to actions. The quality of an action taken by an agent during the RL process is evaluated based on a numerical signal, known as *reward* (or reward signal), received from the environment after the agent having performed such an action. A *reward function* determines the reward an agent will receive from the environment. The *designer*[1] of the RL system is responsible for specifying a reward function which, when optimized by an agent, results in the behavior desired by the designer, in the sense of being a behavior that solves the learning problem/task of interest. The performance of an agent or collective of agents (known as multi-agent RL - MARL) learning via RL is directly related to the reward function being optimized by the agent because the reward function is the mechanism responsible for defining the goal of the learning task.

The use of effective reward functions provides benefits to the agent that go beyond the acquisition of an optimal policy. Multiple reward functions, when optimized, may produce a same optimal policy, but under different learning effort. As an example, consider the episodic learning task illustrated in Figure 1.1 (for details, see Section 5.1). In this task, introduced by Buşoniu, Babuška and Schutter (2010), for agent 1 to solve the task it must:

i) find the object located in the maze;

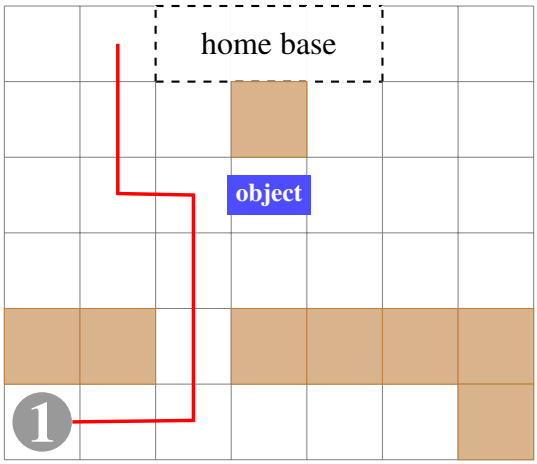
ii) grasp the object; and

iii) transport the object to the home base.

The optimal solution for the task is represented by the red line, which takes 8 steps/movements. The authors of this scenario provide a reward function (Equation 1.1) that when optimized can guide the agent to solve the task at an optimal time. This reward function rewards the agent with a reward signal in the set $\{1.0, 0.1, 0\}$ according to the current state (s) of the agent. However, many more reward functions, such as the one proposed in Equation 1.2, may also guide the learning of that same optimal policy for that task.

[1] Whenever this thesis refers to a "designer", it is not necessarily referring to a single person, but to the team that is developing a reinforcement learning-based solution.

Figure 1.1: Object transportation domain. The learning task consists in agent 1 transport the object to the home base.

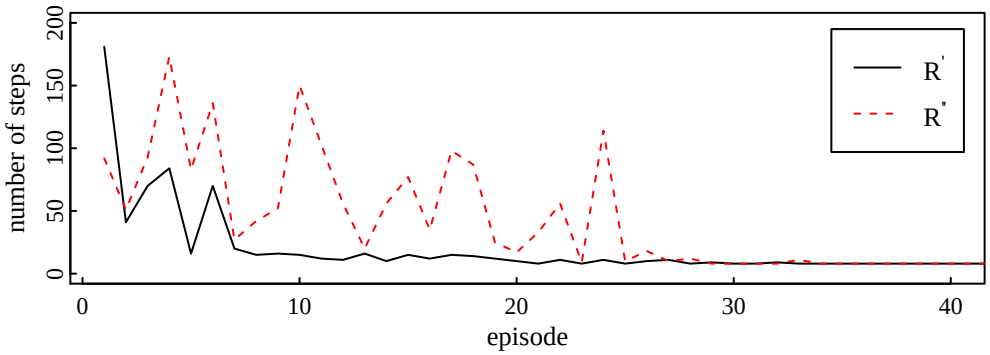


$$R'(s) = \begin{cases} 1, & \text{if the object is at the home base} \\ 0.1, & \text{if the object is grasped} \\ 0, & \text{otherwise} \end{cases} \tag{1.1}$$

$$R''(s) = \begin{cases} 1, & \text{if the object is at the home base} \\ 0, & \text{otherwise} \end{cases} \tag{1.2}$$

In this task, the performance measure for the agent is given by the number of steps it takes to accomplish the task. Figure 1.2 shows the performance of the agent learning with the reward functions R' and R''. In the final episodes, both functions were able to converge to the optimal solution of the task, leading to the conclusion that they are equivalent in finding the optimal policy for this task. However, now, consider a second performance measure, which does not only assess the quality of the learned behavior in the final episode, but also the *learning effort* the agent spend along its lifetime to acquire such behavior. The learning effort is represented by the accumulative amount of steps (or decisions) taken by the agent along the episodes of its lifetime. In case of taking into account the learning effort yielded by the reward functions R' and R'' to acquire the optimal policy, they are no longer equivalent. By learning with the reward function R', the agent takes $\approx 50\%$ fewer steps during its lifetime when compared to R''. From episode 10 to 25, R' had already found near-optimal solutions, while R'' still learns poorer solutions in that period. In other words, for the learning task illustrated in Figure 1.2, it is easier for the agent to learn an optimal policy by being guided by the reward function R' than reward function R''.

Figure 1.2: Performance (# steps) of agent 1 when solving the object transportation problem with reward functions R' and R''.



For a simple scenario like this, it would be more interesting for the designer to use reward function R' instead of R'', since it provides faster learning convergence. However, is R' the most suitable reward function for the task? There is no direct answer to this question because there are many more reward functions that can represent such a learning task. Table 1.1 presents some of the hypothetical situations in which the designer may consider to be interesting to reward the agent. In such case, finding of an adequate reward function is hard due to the space of available reward functions being composed by any combination of these situations and its respective rewards.

Table 1.1: Hypothetical situations in which to reward the agent.

#	Situation
1	if the object is at the home base
2	if the agent grasped the object
3	if the agent hit a wall
4	if the agent tried to go to a cell occupied by an object
5	if the agent chose to stand still

As demonstrated in the object transportation scenario, it is not easy for a designer to define an effective reward function for a given problem. Although it is assumed that the designer of an RL system has enough knowledge to design RL agents, the quality of the behavior learned by such agents is very sensitive to basic choices made by the designer regarding two reward function related design choices:

i) in which situation to reward the agent?

ii) how much to reward the agent in each situation?

Depending on the characteristics of the learning task, other issues may also arise. Single-agent problems usually have fewer issues associated with reward function design than multi-agent problems; in multi-agent problems, e.g., the strategy used by the system's

designer may change according to the type of the learning task—cooperative, competitive or mixed—and the number of learning agents. For instance, in multi-agent cooperative tasks, the use of joint-actions and team rewards may be feasible, as long as the task has few agents. In scenarios with a large number of agents this strategy can suffer from the curse of dimensionality (SUTTON; BARTO, 1998).

The goals of a system's designer in designing a given reward function R is that, when an agent is optimizing R (i.e., learning with reward function R), such agent follows some behavior or solves some task of interest to the system's designer. An inappropriately designed reward function may lead the agent to learn inappropriate behaviors, i.e., a behavior that is not of interest to the designer. Moreover, even though a given reward function can guide the agent to an optimal policy, it is still possible that the learning effort expended—regarding computational resources or time—to learn such a policy renders the use of RL unfeasible. In the modeling stage of a reinforcement learning problem, the system's designer usually adopts the following strategy when creating reward functions:

i) to positively reward actions that lead the agent to desired states; and

ii) to negatively reward actions that lead the agent to undesired states.

Such strategy stimulates behaviors driven by extrinsic motivations, i.e., motivations generated by immediate rewards directly linked to the goals of the agent (BARTO; SINGH; CHENTANEZ, 2004). Singh, Lewis and Barto (2009), Singh et al. (2010) argue that this strategy may not be the best one. The authors show that agents could benefit by rewarding intermediate states because this instigates behaviors driven by curiosity, novelty, surprise and other internally-mediated features that are usually associated with intrinsic rewards.

To design reward functions with such properties, Singh, Lewis and Barto (2009) introduced the *Optimal Reward Problem* (ORP). An ORP is composed of and specified by two functions: i) A reward function R that guides the learning process of the agent; and ii) a fitness function F that evaluates the quality of the learned behavior. The fitness function represents the desire of the system's designer. Solving the ORP consists of finding the *Optimal Reward Function* (ORF), R^*, that maximizes the fitness function. In Singh et al. (2010), the ORP is (approximately) solved through a brute force strategy. Subsequent works (SORG; LEWIS; SINGH, 2010b; NIEKUM; BARTO; SPECTOR, 2010; LIU et al., 2012; LIU et al., 2014) proposed automated and more efficient methods for dealing with ORP in single or multi-agent settings. Such methods present several limitations, which only allow them to be applied in very specific learning tasks.

There are non-ORP strategies that can be used, at least partially, to handle designing reward functions. The technique called *difference rewards*, presented by Tumer and Agogino (2007), stimulates cooperation among a collective of agents through modifications in the basic structure of the reward function. However, its use is limited to cooperative learning tasks were complete environment observation is available. Another well-known type of technique is known as *inverse RL*, which identifies the reward function that produces a behavior that was previously observed by a specialist. Since it requires a set of observations of optimal behaviors, it is not possible to apply inverse RL to many problems. Finally, techniques based on *reward shaping* aim at speeding up the learning convergence by providing domain knowledge in the form of additional reward signals. However, reward function design is not addressed in reward shaping technique.

Each of the methods previously mentioned present their own limitations that prevent them from being applied to other types of learning problems from those they were designed for. The literature has a gap regarding a general method that automates the reward function design in different types of RL/MARL problems. To fill such a gap, the following research question is considered in this thesis:

Research Question.

Is there a method that automates the reward function design process and that can be applied to most types of reinforcement learning tasks?

In this thesis, the idea of an *automated method* refers to a method that can automatically find an adequate reward function for a task, given a set of appropriate inputs provided by the system's designer. When this thesis refers to *types of reinforcement learning tasks*, it is referring to mono and multi-agent tasks and their variants.

Among the existing methods, the one that best matches the research question of this thesis is the ORP because it has already been developed to find the best reward function according to the preferences of the system's designer. Based on the ORP, this thesis seeks to address its research question through the following research hypothesis:

Research Hypothesis.

By providing the necessary modifications to the Optimal Reward Problem, it is

possible to construct an automatic method that can deal with reward function design in most types of reinforcement learning problems.

The present thesis proposes a flexible approach for designing effective reward functions, called the *Extended* Optimal Reward Problem (EORP). The term *flexible* refers to the broad range of single and multi-agent problems in which the proposed approach can be applied. The term *effective* refers to finding reward functions that, when optimized, results in a rapid acquisition of optimal policy.

1.1 Overview of Proposed Extended-Optimal Reward Problem

As the name suggests, the EORP proposed in this thesis is a more complete and versatile version of the ORP, in which the main limitations of the traditional ORP are addressed. The following sections briefly introduce the main features of EORP that address such ORP limitations.

1.1.1 Automatic Reward Feature Selection

The first limitation of existing works is related to the automatic selection and identification of situations (or state features) in which the agent must be rewarded. Existing approaches consider that the designer defines all the situations that may be rewarded. The method only adjusts the reward signal for each situation. However, some problems may present many situations that could potentially contribute to the reward function (see Table 1.1). If the designer chooses an inappropriate set, it can impact the quality of the reward function found by solving the ORP. The proposed approach can identify automatically both situations and reward signals that are relevant for the optimal reward function. The designer should just provide the set of potential situations to reward as input.

1.1.2 Generality of Application

The second limitation of existing works is the lack of generality for dealing with RL problems that may be both single or multi-agent setting. MARL settings were ad-

dressed by Liu et al. (2014). However, their method is limited to common-payoff tasks and model-based algorithms. The approach proposed in this thesis is general for dealing with any MARL problem (including single-agent RL problems) because it uses an evolutionary strategy that is independent of the RL problem and can deal with single/multi-agent tasks that use both model-free and based methods.

1.1.3 Scalability in Multi-Agent Settings

The third limitation of existing works is associated with scalability issues. In Liu et al. (2014), for each learning agent, a private reward function is optimized in order to deal with a specific task. However, in problems with a large number of agents, the optimization may not converge to desired solutions due to the slow convergence of optimization methods in high-dimensional search spaces. The approach presented in this thesis scales because EORP optimizes a unique reward function for all agents that share a common task. Therefore, for a given problem, regardless of the number of agents, the dimensionality of the search problem is always the same.

1.1.4 Learning Effort Evaluation

The fourth limitation of existing approaches is related to the evaluation of an ORF. The space of reward functions may contains multiple reward functions that produce the same behavior, but that differ in learning effort required to acquire an optimal policy. The proposed approach takes into account the trade-off between fitness and learning effort spent in the learning process. Therefore, functions found by the EORP aim at producing the best behavior (fitness) in the best learning time (effort).

1.2 Publications

The scientific contributions that lead to this thesis are:

GRUNITZKI, R.; SILVA, B. C. da; BAZZAN, A. L. C. A flexible approach for designing optimal reward functions. In: DAS, S. et al. (Ed.). **Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Sys-**

tems (AAMAS 2017). São Paulo: IFAAMAS, 2017. p. 1559–1560. Disponível em: <http://ifaamas.org/Proceedings/aamas2017/pdfs/p1559.pdf>.

GRUNITZKI, R.; SILVA, B. C. da; BAZZAN, A. L. C. Towards designing optimal reward functions in multi-agent reinforcement learning problems. In: **Proc. of the 2018 International Joint Conference on Neural Networks (IJCNN 2018)**. Rio de Janeiro: [s.n.], 2018.

In Grunitzki, Silva and Bazzan (2017), the EORP and its evaluation in a coordinated multi-agent object transportation problem is presented. Analysis of the method in a traffic assignment problem, as well as a comparison to a method for reward design in multi-agent settings (namely, difference rewards) is presented in Grunitzki, Silva and Bazzan (2018).

In addition to these works, others have been developed throughout the PhD course that lead to this thesis:

GRUNITZKI, R.; BAZZAN, A. L. C. Comparing two multiagent reinforcement learning approaches for the traffic assignment problem. In: **Intelligent Systems (BRACIS), 2017 Brazilian Conference on**. [S.l.: s.n.], 2017.

BAZZAN, A. L. C.; GRUNITZKI, R. A multiagent reinforcement learning approach to en-route trip building. In: **2016 International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2016. p. 5288–5295.

GRUNITZKI, R.; BAZZAN, A. L. C. Combining car-to-infrastructure communication and multi-agent reinforcement learning in route choice. In: BAZZAN, A. L. C. et al. (Ed.). **Proceedings of the Ninth Workshop on Agents in Traffic and Transportation (ATT-2016)**. New York: CEUR-WS.org, 2016. ISSN 1613-0073. Disponível em: <http://ceur-ws.org/Vol-1678/paper12.pdf>.

GRUNITZKI, R.; RAMOS, G. d. O.; BAZZAN, A. L. C. Uma ferramenta para alocação de tráfego e aprendizagem de rotas em redes viárias. In: **Anais do XXVIII Congresso de Pequisa e Ensino em Transportes (ANPET 2014)**. [s.n.], 2014. ISBN 978-85-87893-17-8. Disponível em: <www.inf.ufrgs.br/maslab/pergamus/pubs/grunitzki+2014-anpet.pdf>.

RAMOS, G. de. O.; GRUNITZKI, R. An improved learning automata approach for the route choice problem. In: KOCH, F.; MENEGUZZI, F.; LAKKARAJU, K.

(Ed.). **Agent Technology for Intelligent Mobile Services and Smart Societies**. [S.l.]: Springer Berlin Heidelberg, 2015, (Communications in Computer and Information Science, v. 498). p. 56–67. ISBN 978-3-662-46240-9.

RAMOS, G. de. O.; GRUNITZKI, R.; BAZZAN, A. L. C. On improving route choice through learning automata. In: **Proceedings of the Fifth International Workshop on Collaborative Agents – Research & Development (CARE 2014)**. [s.n.], 2014. p. 1–12. Disponível em: <http://www.inf.ufrgs.br/maslab/pergamus/pubs/Ramos+2014care.pdf>.

GRUNITZKI, R.; RAMOS, G. d. O.; BAZZAN, A. L. C. Individual versus difference rewards on reinforcement learning for route choice. In: **Intelligent Systems (BRACIS), 2014 Brazilian Conference on**. [S.l.: s.n.], 2014. p. 253–258.

All above publications are in some way related to the research theme of this thesis. In Grunitzki, Ramos and Bazzan (2014a), the use of MARL and difference rewards in traffic assignment problems are investigated. The use of learning automata for dealing with this same problem is addressed in Ramos, Grunitzki and Bazzan (2014), Ramos and Grunitzki (2015). A macroscopic simulator for dealing with traffic assignment in a multiagent perspective is presented in Grunitzki, Ramos and Bazzan (2014b). Grunitzki and Bazzan (2016) investigate the use of car-to-car communication in a MARL setting for the traffic assignment problem. A strategy in which the agents build the route along the trip is investigated in Bazzan and Grunitzki (2016). This strategy is compared to a strategy in which the agents learn to select a route from a set of pre-computed routes in Grunitzki and Bazzan (2017).

1.3 Organization of the Chapters

The rest of this thesis is organized as follows. Chapter 2 introduces the basic concepts of multi-agent systems and multi-agent reinforcement learning. The related work on reward function design is presented and discussed in Section 3. The proposed approach appears in Chapter. 4. Chapter 5 performs experimental evaluations of the EORP. The conclusions and future work are presented in Chapter 6.

2 BACKGROUND

This chapter introduces the basic concepts of autonomous agents and multi-agent systems (Section 2.1), machine learning (Section 2.2), and reinforcement learning and its main elements (Section 2.3).

2.1 Autonomous Agents and Multi-Agent Systems

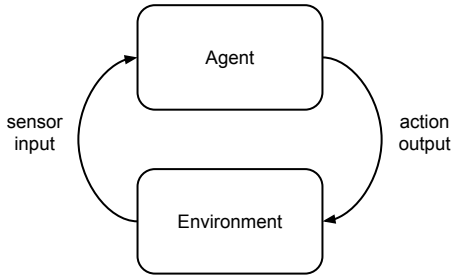
There is no canonical definition for the term *agent* accepted by the whole community: there are many debates and controversies regarding the various definitions presented in the literature. Despite this, the different definitions—such as the ones presented by Jennings (2000) and Wooldridge (2009)—have the term "autonomy" as a central concept underlying the idea of an agent. According to Wooldridge (2009), this difficulty of definition is associated with the fact that various attributes of the agent are of different importance and change as a function of the domain. This is understandable since in some applications the ability to learn behaviour from experiences is of fundamental importance, but for others, learning is undesirable. In many domains, agent's behaviors can be pre-programmed. However, some tasks are so complex that programming behaviors a priori become impractical. In such situations, would be interesting to provide to agents the ability to *learn*[1] a solution autonomously. Given this scenario, this thesis adopts the following concept of agents:

> *An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives* (WOOLDRIDGE, 2009).

This interaction process between agent and environment is illustrated in Figure 2.1. Note that an agent situated in its environment takes sensory inputs from the environment, and produces as output actions that affect the environment. In most environments, an agent will not have full control over its environment. That means that the same action taken twice in apparently identical circumstances may have different effects. This problem becomes even harder when multiple agents are interacting in the same environment. In such multi-agent systems, the outcomes of an agent's action are influenced by the environmental noise and also by the actions performed by the other agents.

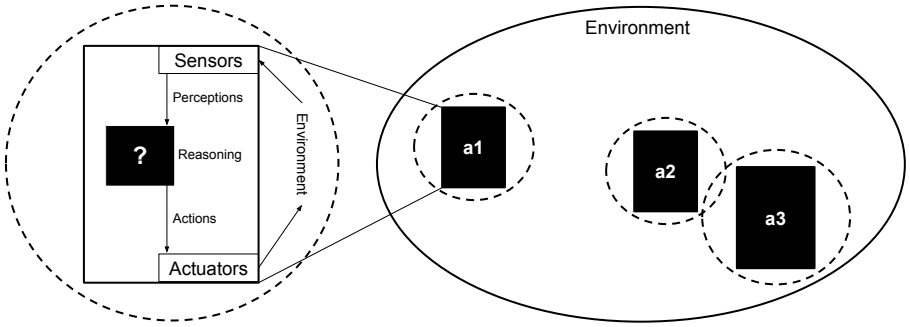
[1] When the agent has a model about the environment dynamics, it can also plan over such model to find a solution (policy). Thus, in some cases, the use of *planning* methods would also be practical. However, planning is not covered in this thesis.

Figure 2.1: Agent-environment interaction schema.



The typical structure of a *multi-agent system*, suggested by Jennings (2000), is illustrated in Figure 2.2. The system is composed of multiple agents that can act in the environment and interact with one another through communication. The agents have their own *sphere of influence*, meaning that they will have control over different parts of the environment. These spheres of influence may have intersections that give rise to dependency relationships between agents. For the schema in Figure 2.2, consider that agent a_2 and a_3 are trying to cross a door that is too narrow for both to pass simultaneously. The relationship between these two agents could be a *power relationship*, where one agent is the *boss* of another, and such boss decides who must pass first (WOOLDRIDGE, 2009).

Figure 2.2: Typical structure of a multi-agent system.



The following sections introduces a few techniques.

2.2 Machine Learning

Machine learning is the field of computer science that gives the ability to learn with data, without being beneficially programmed. In other words, machine learning progressively improve the performance of a specific task. Russell and Norvig (2010) classify the type of the learning in three categories:

i) unsupervised learning;

ii) supervised learning; and

iii) reinforcement learning.

In unsupervised learning, the agent learns patterns in the input even though no explicit feedback is supplied. The most common unsupervised task is clustering. In supervised learning, the agent has access to some examples of input-output pairs and learns a function that maps inputs to outputs. The most common supervised tasks are classification and regression. In reinforcement learning, the agent acquires information about its current state, making reinforcement learning somewhat related to supervised learning. However, it is different from supervised learning because the information acquired by a reinforcement learning agent is not the correct ground truth label on action to be performed at that state.

The focus of this work is on reinforcement learning. For this reason, supervised and unsupervised learning will not be discussed in detail in this thesis. In the next section, reinforcement learning is detailed.

2.3 Reinforcement Learning

Reinforcement learning deals with the problem of an agent learning a behavior to accomplish a task through successive interactions with the environment (SUTTON; BARTO, 1998). Such behavior involves the mapping of situations (states) to actions, in a way that maximizes some numerical utility or reward. Unlike other machine learning methods, an RL agent is not told which action to take in each situation. Rather, it must discover which actions maximize its cumulative reward by trying them out.

On single-agent RL tasks, the reward signal received by the agent is influenced only by the consequences of its actions and some noise due to the environment dynamics. However, with multiple agents, the learning task becomes more difficult because the actions of other agents also influence the reward received by an agent. Multiagent RL techniques emerged to handle these types of problems. Approaches such as joint-action learning (CLAUS; BOUTILIER, 1998), nash Q-Learning (HU; WELLMAN, 2003), and gradient ascent algorithms (GRUNITZKI; RAMOS; BAZZAN, 2014a) were initially introduced as proper solutions for MARL problems. Although such approaches present convergence guarantees to optimal solutions[2], their application is limited to a small set of

[2]Given some particular conditions of each algorithm.

problems such as games of few agents (normally only two) and few actions.

In the *multiple independent learners* (MIL) strategy, the agents' decision-making process are implemented in an individual manner, i.e., joint-actions are not considered (CLAUS; BOUTILIER, 1998). This way, an agent understands the learning and behavioral changing of the other agents as a changing in environment dynamic. The agent's decision-making process in this setting, and also in single agent reinforcement learning, can be modeled as a *Markov decision-process* (MDP). An MDP is a tuple $< S, A, T, R >$, where:

- S is the set of environment states where the agent may be situated in;
- A is the set of actions that the agent can execute. $A(s)$ is the set of available actions at a specific state s;
- $T : S \times A \leftarrow \Pi(S)$ is the transition function, where $\Pi(S)$ is a probability distribution over S; and
- $R : S \times A \rightarrow \Re$ is the reward function that returns a reward $R(s, a) = r$ for taking action $a \in \mathcal{A}(s)$ in state $s \in S$.

The objective of an RL agent is to find a policy $\pi : S \rightarrow A$ that maximizes its expected cumulative reward. To maximize the reward received throughout all interactions, the agent must select each action according to a strategy that balances exploration (gaining of knowledge) and exploitation (usage of knowledge). A well-known exploration strategy is the ϵ-decreasing, which consists in choosing random actions (exploration) with probability $\epsilon \in [0, 1]$ or choosing best actions (exploitation) with probability $1 - \epsilon$. This way, in the beginning, ϵ has a high exploration, and decreases exponentially along the episodes, as in:

$$\epsilon_\lambda = \epsilon \times (\Delta\epsilon)^\lambda \tag{2.1}$$

where $\Delta\epsilon$ represents a decay rate in the interval $[0, 1]$; and $\lambda \in \Lambda$ is an episode.

Value functions are functions of states (or of state-action pairs) that estimate how good is for the agent to be in a state. The notion of "how good" is given is terms of future reward that can be expected. The Q-learning algorithm, proposed by Watkins and Dayan (1992), is a traditional algorithm used to learn value functions independently to the policy being followed. Its update rule is shown in Equation 2.2, where $< s, a, s', r >$ is an experience tuple, meaning that the agent performed action a in state s, reaching s', and

receiving reward r. Action a' is one of the possible actions on s', $\alpha \in (0, 1]$ is the learning rate, and $\gamma \in (0, 1]$ is the discount factor. $Q(s, a)$ is an entry indexed by state s and action a in the MDP, which stores the value functions (or Q-values) of each state-action pair. The Q-value $Q(s, a)$ is the expected discounted return for executing action a at state s and following the policy π thereafter.

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') \right] \tag{2.2}$$

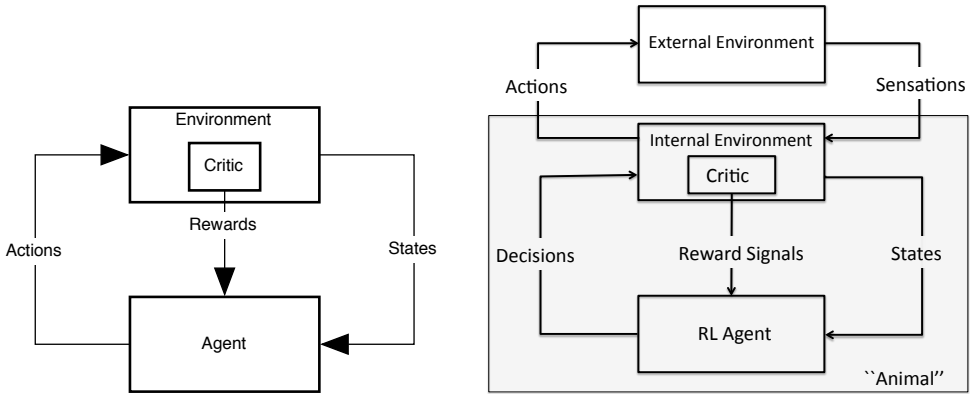
3 REWARD FUNCTION DESIGN

A reward function R is the RL element that defines and represents a specific learning task (NG; HARADA; RUSSELL, 1999). The system's designer is responsible for setting such function that guides the learning process. There is no general rule for the definition of a reward function. In practice, the system's designer defines a reward function empirically, based on his or her intuition about the answers for the questions "*when to reward the agent?*" and "*by how much to reward the agent?*". The literature presents methods that support, at least partially, the system's designer in such a task. The following sections present the main methods related to reward function design. At the end of this chapter (in Section 3.5), a general discussion about such methods is provided.

3.1 Optimal Reward Problem

In the standard view of the reinforcement learning, the agent's behavior is evaluated through reward signals, which are *external* from the agent. A *critic* transforms the actions of the agent in reward signals, as illustrated in the left panel of Figure 3.1. However, this traditional RL framework should not be considered to encompass an entire *animal* or agent. The work of Sutton and Barto (1998) points out that an RL agent represents the learning component of the brain. The remaining physical components are separated from the learning component but are still part of the broader *agent animal*. In Barto, Singh and Chentanez (2004), the classical RL framework is extended according to this concept, as illustrated in the right panel of Figure 3.1. In particular, the environment is factored into two components: the *internal* environment and the *external* environment. The animal takes actions in the physical word, for example, by applying torque to its arms, while the RL agent itself makes decisions, such as the decision to move in a particular direction. The internal environment is responsible for physically sensing the environment and converting those sensations into signals that the RL agent can interpret. A designer is typically responsible for specifying the entire animal, including its internal environment. In practice, however, the existence of an external critic in the environment to send the appropriate reward signals to the agent is not common. Instead, a designer builds an autonomous critic. Furthermore, the interpretation of any external signal as a reward signal is part of agent design. Thus the critic is built along with the other components of the agent, and it is better considered to be a part of the animal.

Figure 3.1: Agent-environment interactions; adapted from Barto, Singh and Chentanez (2004). Left panel: traditional RL framework. Right panel: intrinsically motivated RL framework.



In the classical RL framework there is only one function (the reward function) which represents two purposes:

1. defining the preferences of agent's designer;

2. guiding the behavior of the agent.

The use of a single reward function confounds these two purposes (BARTO; SINGH; CHENTANEZ, 2004; SINGH; BARTO; CHENTANEZ, 2004). By definition, this standard RL framework assumes that the goals of both agent and designer should be the same. For example, in a hypothetical learning task in which a dog is trained to shake hands, it receives food every time it accomplishes the task—that is, shaking hands with its owner. The *fitness* of the owner is related to the accomplishment of the shaking hand task, while the *reward* given to the agent is the food. The owner's goal is to maximize the number of shaking hands. On the contrary, the dog's goal is to maximize the amount of food received. Although there is a synergy between reward and fitness functions, one does not necessarily represent the objective of the other. In this analogy, the dog does not want to maximize the number of shaking hands, and the owner does not want to maximize the given food. However, the synergy between both enables the correct learning of this behavior.

In Singh, Lewis and Barto (2009), Sorg, Lewis and Singh (2010b), the authors suggest that both reward function and fitness function must be separated. During the modeling stage of an RL problem, it is assumed that the designer knows the *fitness function* because such function measures what the designer expects the agent to do. However, the correct definition of *reward functions* is not so direct in complex problems. The authors present the *Optimal Reward Problem* (ORP), that is, the problem of finding an *Optimal*

Reward Function (ORF) that, when maximized by the agent, also maximizes the fitness function.

The ORP is formally defined as follows. At each time step, an agent $\mathcal{G}$ receives an observation $o \in \mathcal{O}$ from its environment $\mathcal{M}$, takes action $a \in A$, produces a history h, and repeats this process for a certain time horizon. A reward function R represents the agent's goals. The designer's goals are represented by a fitness function F, which produces a cumulative return, $F_R(h)$ for a reward function R over a history h. The ORF is given in Equation 3.1, where R^* is an ORF that maximizes the designer's expected fitness F of an agent $\mathcal{G}$ in some environment $\mathcal{M}$, and the maximization is over the set of reward functions $\mathcal{R}$. The expectation operator is denoted by $\mathbb{E}$. The notation $h \sim \mathcal{M}\langle\mathcal{G}(R)\rangle$ denotes that h is a sample history generated when agent $\mathcal{G}$ acts in environment $\mathcal{M}$, using reward function R.

$$R^* = \arg\max_{R \in \mathcal{R}} \mathbb{E}\left[F_R(h) \,|h \sim \mathcal{M}\langle\mathcal{G}(R)\rangle\right] \tag{3.1}$$

This ORP formulation presents minimal assumptions about the agent and comes with a simple guarantee: an ORF performs at least as well as the objective reward function—the conventional choice—as long as the objective reward function is in the search set $\mathcal{R}$ (SORG, 2011). This guarantee holds whether or not the reward function of the agent and the fitness function of the designer are the same (SORG; LEWIS; SINGH, 2010a). By including the constraint that $R \in \mathcal{R}$, the proof is straightforward.

In practice, it is not easy to exactly find R^*. It must be approximated, as done in existing works. The work of Singh, Lewis and Barto (2009), Singh et al. (2010) focuses on the theoretical formulation of ORP. For solving it, a brute force strategy is used, which consists of the discretization of the space of reward functions. The following works on ORP introduce methods to find approximate solutions to the ORP. In Niekum, Barto and Spector (2010), a genetic programming method is proposed. Sorg, Lewis and Singh (2010b) propose a gradient-based method for online solving of ORP. Both approaches perform well in finding approximated solutions for the problem when compared against a force brute strategy. Although there is no direct comparison of these methods, the main difference between them is that the gradient-based updates the reward function during the learning process, while the genetic programming-based needs to test a given reward function in a complete learning process to adjust it. In theory, the gradient-based may be faster than the genetic programming method. However, experimentation must be performed to support this hypothesis.

All works discussed so far were developed for single-agent RL problems. The first work to deal with the ORP in a multi-agent perspective is the one presented by Liu et al. (2014). In this work, the algorithm proposed by Sorg, Lewis and Singh (2010b) is extended for multi-agent cooperative settings (specifically, common-payoff or team). In this multi-agent ORP, the team reward function is used as the fitness function. Each agent learns an individual reward function that guides its behavior. The obtained results showed that this approach could improve the performance of the agents when compared to traditional approaches that use only the shared team-reward function. The explanation for this is that the learning of individual rewards stimulates the specialization of the agents in different roles.

The approach of (LIU et al., 2014) assumes that each agent has a perfect model of the dynamics of the environment and use upper confidence bound on trees (UCT) algorithm to plan. Agents do not know which action other agents will take, but they can observe how other agents have acted in the past. The current state of other agents is available for all agents. During the learning process, each agent updates its environment model based on the actions performed by the other agents. These actions are used in the proposed architecture during the joint-action planning. The actions of other agents are estimated based on probability distributions acquired from the past.

The original work of (SINGH; LEWIS; BARTO, 2009) inspired all ORP-based methods presented so far. Each of them proposes some improvements and modifications to the original ORP structure to allow for its application in some specific task. Table 3.1 provides a comparison of such approaches regarding:

- solver: the algorithm used to solve the ORP;
- technique: refers to the applicability of the method to learning or planning settings;
- single-agent: applicability to single-agent problems;
- multi-agent: applicability to multi-agent problems;
- RF update: refers to the strategy adopted to update/evolve the reward function (RF) during the optimization process. Consider that when an agent is born/created it receives a reward function, which can be updated on or off-line. *On-line* update means that the structure of the reward function changes while it is in use by the agent(s) in its(their) lifetime. *Off-line* update means that the structure of the reward function does not chance while the agent(s) is(are) using it;

- reward scope: represents the *scope* in which the reward signal is propagated to the agents. *Individual* scope represents the problems in which each agent receives private reward signals after executing its actions. Different from *team* scope, where the entire team of agents is rewarded with the same reward signal after they have executed their actions.

- environment observation: refers to the nature of environment information used by the reward function. *Partial* environment observation represents the information the agent can observe in its current state. *Complete* environment observation represents all the information present in the environment, even if such environment information can not be physically observable by the agent from its current state.

- task type: could be *competitive* and/or *cooperative* in the multi-agent case.

Table 3.1: Relevant aspects of ORP-based approaches.

Aspect	**Singh et al. (2010)**	**Niekum et al. (2010)**	**Sorg et al. (2010)**	**Liu et al. (2014)**
solver	brute force	genetic programming	gradient ascent	gradient ascent
technique	learning	learning	planning	planning
single-agent	✓	✓	✓	✗
multi-agent	✗	✗	✗	✓
RF update	off-line	off-line	on-line	on-line
reward scope	individual	individual	individual	team
env. observation	partial	partial	complete	complete
task type	competitive competitive	 cooperative		

The existing ORP techniques encompass a not-so-narrow range of application. However, for each type of problem, it is necessary to use a different technique. For example, the work of Liu et al. (2014) applies to multi-agent cooperative tasks, but it is not applicable to single-agent or planning tasks. Besides this, there are some gaps in the application of these methods. For instance, no method deals with learning technique in multi-agent settings. Another gap is the application in multi-agent competitive settings.

3.2 Reward Shaping

In RL, the value functions are usually initialized with random values, i.e., optimistic or pessimistic expectations of reward to be received in each state-action pair. During the modeling phase of RL agents, the designer often has knowledge about the learning domain that can be given to a learning agent during its learning process. Knowledge-based RL mechanisms such as potential-based reward shaping (PBRS) (NG; HARADA;

RUSSELL, 1999; WIEWIORA; COTTRELL; ELKAN, 2003; DEVLIN et al., 2014) deal with such incorporation of domain knowledge into RL agents, aiming at guiding domain exploration. The major advantage of such knowledge incorporation is the possibility of a reduction in the number of sub-optimal decisions taken by the agent and, consequently, an alleviation in the impact of the explosion of the state-action space.

A potential function maps states to potential numerical values that represent the designer's preference of agent being in a given state. For instance, it is a common strategy to define high potential values to goal states and, this way, decrement the potentials linearly according to the distance of other states to the goal. This strategy encourages the agent to visit the states more related to the goal. Domain knowledge can be represented by the PBRS function $F(s, s')$ defined in Equation 3.2, which is given by the difference between: i) a potential function of next state, $\Phi(s')$, discounted by γ; and ii) a potential function of the current state, $\Phi(s)$.

$$F(s, s') = \gamma\Phi(s') - \Phi(s) \tag{3.2}$$

The use of PBRS does not require major modifications to the standard RL framework. In a traditional algorithm such as Q-learning, the incorporation of PBRS in its update rule is given by Equation 3.3. Note that when an agent takes an action a in state s and transits to state s', it receives an extra reward signal given by $F(s, s')$ that is summed with the one provided by the reward function $R(s, a)$.

$$Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha \left[R(s, a) + F(s, s') + \gamma \max_{a'} Q(s', a') \right] \tag{3.3}$$

The work of Ng, Harada and Russell (1999) proves that the policy acquired by an agent learning with PBRS is equivalent to the one obtained by an agent learning without PBRS. Further work by Wiewiora, Cottrell and Elkan (2003) demonstrates that the use of PBRS without value functions initialization is equivalent to the use of value functions initialization without PBRS. However, Devlin and Kudenko (2012) prove that such a claim only holds when the PBRS function is deterministic.

Note that PBRS was developed for the purpose of accelerating learning in a particular environment. The kind of reward provided by PBRS has been explicitly developed with the intention of not changing the long-term behavior of an agent. On the other hand, ORP is conceptually different. The work of Sorg, Lewis and Singh (2010b) shows that it

is sometimes beneficial to modify the long-term behavior of an agent. ORP can be used to improve the performance of agents. Besides this, PBRS does not handle with the problem of finding good reward functions. Reward shaping assumes an existing reward function and focuses on providing extra knowledge in the form of addictive reward signals that do not change the optimal policy of the agent.

3.3 Difference Rewards

In multi-agent cooperative problems, it is normal to design reward functions that provide rewards based on a system's performance measure of the problem. However, according to Tumer and Agogino (2007), in some domains, such reward structure leads to slow learning. Given that reinforcement learning agents aim to maximize their individual rewards, a critical task is to create "good" agents' rewards, or rewards that when pursued by all the agents of a RL system lead to good system's performance. The system's performance can be measured by a system's utility function $G(z)$, where z is a variable that represents the state-action pairs of all agents in the system.

Difference Rewards are functions proposed by Tumer and Agogino (2007), which aim at providing reward signals that are both sensitive to the agents' actions and aligned with the system's performance. Consider the difference rewards as in Equation 3.4, where z_{-i} is the state-action pairs for a theoretical system without the contribution of agent i. All components of z that are affected by agent i are replaced with the fixed constant c_i.

$$D_i \equiv G(z) - G(z_{-i} + c_i) \tag{3.4}$$

Using a $null$ vector in c_i is equivalent to taking agent i out of the system. Intuitively this causes the second term of the difference rewards to evaluate the performance of the system without i and, therefore, D evaluates the agent's contribution to the system's performance. According to Tumer and Agogino (2007), there are two advantages using Equation 3.4 for rewarding:

1. the second term removes a significant portion of the impact of the other agents in the system and provides an agent with a "cleaner" signal than G. This benefit has been dubbed *learnability* in previous works (AGOGINO; TUMER, 2005; TUMER; WOLPERT, 2004);

2. the second term does not depend on the actions of agent i. In other words, any

action taken by agent i that improves D also improves G. This term measures the *alignment* between two rewards. It has been dubbed *factoredness* in previous works (AGOGINO; TUMER, 2005; TUMER; WOLPERT, 2004).

Difference rewards can be applied to any linear or nonlinear system's utility functions. However, its effectiveness depends on the domain, and on the interaction among the agents' utility function (TUMER; AGOGINO, 2007).

3.4 Inverse Reinforcement Learning

Inverse reinforcement learning deals with the problem of identifying the reward function that can explain the observed behavior of an agent. Consider the simple case where the state space is finite, the environment model is known, and the agent's policy is observed. Formally, given a finite state space S, a set of k actions A, transition function T, a discount factor γ, and a policy π; the problem consists in finding the set of reward functions $R^+ \in \mathcal{R}$, where $\mathcal{R}$ is the space of reward functions, such that π is an optimal policy for the MDP $< S, A, T, \gamma, R^+ >$.

IRL is considered as a reverse procedure of RL because it assumes that the expert's demonstration is generated by an optimal policy π^*, generated according to some unknown reward function R^+. The objective of IRL is to learn this unknown reward function.

The original algorithms for IRL formulate the problem as a linear programming procedure with constraints corresponding to the optimal condition. According to Zhifei and Er Meng Joo (2012), three settings (presented below) are the most common in IRL problem formulations. Among these, the last one is the closest to practical problems because, usually, only the expert's demonstrations are available rather than a specific policy.

1. finite-state MDP with known optimal policy;
2. infinite-state MDP with known optimal policy; and
3. infinite-state MDP with unknown optimal policy, but demonstrations are given.

There are several domains in which IRL has been applied, from which it is important to highlight: simulated highway driving (ABBEEL; NG, 2004; SYED; SCHAPIRE,

2008), aerial imagery based navigation (RATLIFF; BAGNELL; ZINKEVICH, 2006), parking lot navigation (ABBEEL et al., 2008), urban navigation (ZIEBART et al., 2008), human path planning (MOMBAUR; TRUONG; LAUMOND, 2009) and quadruped locomotion (KOLTER; ABBEEL; NG, 2008). Problems such as these are abundant in experts demonstrations, which makes it possible to apply IRL. Unfortunately, this is not always the case, because in many problems the behavior that represents the optimal policy is not known. In such situations, the application of IRL is unfeasible.

3.5 Discussion

This thesis proposes a general method for finding effective reward functions, i.e., a method that applies to a wide range of RL/MARL tasks. None of existing methods completely meets the requirements of this thesis because they were developed for very specific purposes, in particular:

- reward shaping can speed-up the RL process by providing extra rewards to agents, but does not deal with the task of designing the reward functions.
- difference rewards makes several assumptions about the use of global information for solving cooperative problems, in multi-agent settings.
- inverse RL is quite versatile about its application, but it is not always possible to obtain a set of expert demonstrations it requires.
- existing ORP-based approaches are not general enough to deal with the diversity of existing RL/MARL tasks, such as multi-agent non-cooperative tasks

Among the above methods, the ORP-based ones best fit the reward function design goals of this thesis. For this reason, the mathematical formulation for finding optimal reward functions proposed in this thesis is strongly inspired in the traditional ORP and its variants.

4 EXTENDED OPTIMAL REWARD PROBLEM

This thesis proposes a new formulation to design reward functions and fill some of the gaps in the field. As mentioned in Chapter 3, none of the existing methods for reward function designing is general enough to handle different settings. Instead, they are aimed at designing reward functions for particular kinds of problems. This thesis seeks to put together the best of each existing method in a single one, allowing it to find *optimal reward functions* that, when optimized, results in behavior expected by the system's designer in the most diverse type of problems.

The term "optimal" concerts to the EORP formulation, which is optimal. None of the arguments of such formulation depends on the search procedure (solver). Finding the true globally-optimal reward functions depends on the solver used to solve the EORP, which may not be able to provide guarantees of finding globally-optimal reward functions. Besides this, this thesis is concerned with the reward functions that confer advantages over others and not with absolute optimality. Similarly, the fact that optimization is at the core of the RL framework does not imply that what an RL system learns is optimal. What matters is the process of improving, not the final results.

The formulation proposed in this thesis is called Extended-Optimal Reward Problem (EORP) because it is strongly inspired by the traditional ORP (defined in Equation 3.1, Section 3.1). The main characteristics of EORP are summarized below.

i) Automatic selection of both features and reward signals that compose the ORF;

ii) Generality of application for dealing with both single and multi-agent problems;

iii) Scalability to problems with a large number of agents; and

iv) Ability to find ORFs that speed up convergence.

Throughout this chapter, it will be detailed how EORP handles each of these characteristics. The rest of this chapter is organized as follows. Section 4.1 presents the mathematical formulation of EORP. The evaluation function is presented in Section 4.2. The reward design space is presented in Section 4.3. Section 4.4 presents a EORP solver.

4.1 Mathematical Formulation

The EORP is given by Equation 4.1, where H is the set of all histories generated by the agents $i \in I$ learning with a reward function R. The two major modifications in this formulation with respect to the ORP, are the introduction of:

i) a reward design space, $\mathcal{R}(J)$, where J is the of reward features defined by the system's designer; and

ii) a function $\mathcal{F}$, called the *evaluation* function, which extends fitness functions by allowing for multiple designer objectives.

These two functions are discussed in details in following sections.

$$R^* = \arg\max_{R \in \mathcal{R}(J)} \mathbb{E}\left[\mathcal{F}_R(H) \,|H \sim \mathcal{M}\langle I(R)\rangle\right] \tag{4.1}$$

4.2 Evaluation Function $\mathcal{F}$

As seen in Chapter 3, Equation 3.1, a fitness function F expresses the goal of the designer through a real scalar. By maximizing F, agents improve their behavior. In the space of reward functions, more than one function $R \in \mathcal{R}$ can produce the same fitness F, but under different *learning effort*. A very intuitive example of this behavior is presented in the object transportation domain (Figure 1.1) on page 14. For the designer, it is interesting to identify the reward function that, when optimized, causes the agent to learn to solve a task *more rapidly*. Another situation that can arise is when another reward function $R' \in \mathcal{R}$ produces a fitness value slightly worse than R, but with lower learning effort. In situations like this, it may be attractive to the designer to give up a bit of fitness for better learning effort. Recall that the traditional ORP does not assist the designer in these decisions because it is only aimed at maximizing fitness.

In the EORP framework, it is considered that the designer may have multiple goals to be maximized, such as fitness and learning effort. Equation 4.1 introduces a multi-objective evaluation function $\mathcal{F}$, which evaluates the n goals produced by H and returns a n-dimensional vector. Its general formulation is given in Equation 4.2:

$$\mathcal{F}(H) = [f_1(H), \ldots, f_n(H)] \tag{4.2}$$

where $\{f_1, \ldots, f_n\}$ are goal functions provided by the designer of the system; a goal function $f : H \to \mathbb{R}$ maps the quality of a set of histories H into a numerical signal.

Consider the example of the evaluation function in Equation 4.3, which is composed by two goal functions: fitness (f_1) and learning effort (f_2). The fitness function measures the quality of the final behavior learned by the agents, while the learning effort function evaluates the amount of effort they spent (e.g. time until convergence) during their learning period. These functions represent the desire of some designer for a given learning task. By solving an EORP that uses such an evaluation function, the ORFs yielded are the ones that produce a high fitness at a low learning effort.

$$\mathcal{F}(H) = [f_1(H), -f_2(H)] \tag{4.3}$$

To model a fitness function, consider the following notation. At each episode λ, each agent $i \in I$ learns a policy that optimizes a given reward function $R \in \mathcal{R}$, and in the process generates a history h_i. Consider $g_1(h_i)$ the fitness produced by i over its history h_i. For MARL tasks, consider the average fitness produced by all agents, as in Equation 4.4.

$$f_1(H) = \frac{\sum_{i \in I} g_1(h_i)}{|I|} \tag{4.4}$$

The effort function for an agent i is given by $g_2(h_i)$. For a collective of agents I, consider the average effort produced by all agents, as in Equation 4.5.

$$f_2(H) = \frac{\sum_{i \in I} g_2(h_i)}{|I|} \tag{4.5}$$

By inserting multiple goals, such as f_1 and f_2, the optimization problem turns into a multi-objective optimization problem. The solution for this is a set of Pareto-optimal solutions[1] (ORFs) that considers the trade-off between possibly conflicting objectives.

The formulation in this thesis considers that each goal function is equally important to the designer. For this reason, it was opted not to use techniques for converting the multi-objective optimization problem into a single objective optimization problem—which is much simpler to solve. Techniques such as assigning weights or providing some ranking order for prioritizing goals (and then convert it to a single-objective optimization

[1] A solution is called Pareto-optimal if none of the objective functions can be improved in value without degrading some of the other objective values. Without additional subjective preference information, all Pareto-optimal solutions are considered equally good (as vectors cannot be ordered completely).

problem) can be applied in the EORP. However, that has the drawback that it loses information about how good one particular objective function is with respect to the others. The EORP is not modeled as a single-objective problem to allow the designer to more easily quantify the trade-offs in satisfying the different goal functions.

4.3 Reward Design Space $\mathcal{R}(J)$

The reward design space $\mathcal{R}(J)$ represents the set of all possible reward functions spanned by a given set of reward features J. A reward feature can be seen as a situation (or state feature) in which the agent may be rewarded. In approaches such as Singh et al. (2010), Niekum, Barto and Spector (2010), Sorg, Lewis and Singh (2010b), Liu et al. (2014), the designer defines a fixed set of features that will compose the reward function, and the ORP solver only adjusts the reward signal of each reward feature. However, a reward feature can be represented at different levels of abstractions. For instance, imagine a maze problem in which an agent starts at a random position and must reach an exit door. During its movement, it may face three kinds of obstacles: trees, cliffs, and walls. Three reward features that indicate situations in which the agent may be rewarded in this task can be postulated:

1. reaching the exit door;
2. reaching an obstacle; and
3. otherwise.

Each type of obstacle is associated with different possible consequences to the agent:

- if it falls in a cliff, it dies and a new episode starts;
- if it hits a tree, it does not move in that time step.

Both obstacles are represented with the single reward feature “reaching an obstacle”. However, it could be interesting to use different rewards signals to differentiate the situations of hitting a tree and falling in a cliff. This could be achieved by adding extra reward features to the reward design space $\mathcal{R}(J)$:

1. reaching the exit door;
2. reaching a three;

3. falling in a cliff; and

4. otherwise.

Note that now the reward feature "reaching an obstacle" was decomposed into two reward features that better specify the type of obstacle the agent is facing.

In problems like this, the decision about which reward features to include in the reward function is a hard one. This becomes even harder with the increase in the number of reward features available in the learning problem. The choice of reward features that indicates that the agent will reach an goal state (e.g., reward feature 1) is more direct for the designer. However, the problem may present several other reward features related to intermediate states (e.g., reward features 2, 3 and 4), in which their relevance is not so direct. By using the EORP, the designer must only define the set of potential reward features. Solving the EORP will automatically discover the most appropriate combinations to be used in the reward function.

In EORP, the designer defines the set of potential situations to reward and the method selects the ones that compose an optimal reward function as well as the respective reward weights of each feature. The potential reward features are *identified* by the EORP because in many problems there are costs associated with the use of each of them. For instance, in the robotic example provided before, the use of particular reward features could be associated with the monetary cost of sensors and its energy consumption costs to identify if a given obstacle is a tree or a wall. By just considering reward functions that use all potential reward features provided by the designer and assigning a zero reward weight to the non-used ones, it results in robotic agents that spend too much energy and costs too much money. This strategy (of selecting reward features) is similar to what is done, e.g., in most works on feature selection: a designer pre-defines a set of potentially useful features for a regressor, and a method selects amongst them. Note that when manually specifying a reward function (as is done in standard RL settings), a designer also needs to pre-define and specify reward features—this is not a bottleneck particular to the EORP. Thus, this thesis does not address the problem of automatic *extraction* of reward features from the environment. It considers that the designer always has enough knowledge to define a proper set of *potential* features.

Formally, the set of hypothetical reward functions J is composed by reward features $j \in J$ that a system's designer considers that can be included in a reward function.

A general reward function $R \in \mathcal{R}(J)$ is given as in:

$$R = \sum_{j \in J} s(j)\, w(j)\, P(j) \tag{4.6}$$

where for each feature j:

- $s(j) \in \{0, 1\}$ indicates whether the situation j is activate/active in state s;
- $w(j) \in \mathbb{R}$ and $-1 \leq w(j) \leq 1$ represents the contribution to the reward signal of reward feature j; and
- $P(j) = \{x \in \mathbb{N} \mid 0 \leq x \leq 1\}$ is an indicator function reflecting whether or not j is used to compose R.

Consider the hypothetical search space in:

$$\mathcal{R}(J) \subset \mathcal{R}^{2|J|} = \left[w(j_1), \ldots, w\left(j_{|J|}\right), P(j_1), \ldots, P\left(j_{|J|}\right)\right] \tag{4.7}$$

This search space contains $|J|$ indicator function $P(j)$ and $|J|$ reward signals $w(j)$. The number of decision variables of an EORP with such a search space is $2|J|$. The EORP optimizes a single reward function that is used by the set I of learning agents. This way, independently of the amount of agents in the learning problem, the dimensionality D of the optimization problem is always given by the number of reward features in J, such that $D(\mathcal{R}(J)) = 2|J|$.

4.4 EORP Solver

Having defined the reward design space $\mathcal{R}$ and the multi-objective evaluation function $\mathcal{F}$, the next step is to solve the EORP by finding the set of Pareto-optimal solutions, $R^* \in \mathcal{R}(J)$, that maximize $\mathcal{F}$. Any multi-objective optimization algorithm that deals with real and integer decision variables can be selected in an agnostic manner with respect to the learning problem. The search method is completely independent of the learning problem. Therefore, the EORF criterion can be applied to model-free/based RL problems. This optimization produces only one EORP solution per set of agents so that every agent will learn a policy by maximizing such a function.

In this thesis, it was opted to use a multi-objective version of the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) (IGEL; HANSEN; ROTH, 2007). This

method creates a population of candidate solutions (reward functions) that are evaluated and evolved according to their resulting value of $\mathcal{F}$. Each candidate is evaluated on the learning problem at hand. The method stops when a stopping criterion is satisfied. This thesis considers the CMA-ES stops you after a maximum number of evaluation of the objective function.

Several other multi-objective solvers could be used. Genetic Algorithms (FastPGA, NSGAII, NSGAIII) and multi-objective particle swarm optimization are a few examples. In this thesis, it was opted for using CMA-ES because it has been shown successful in solving many RL applications. As it was capable of identifying solutions close to the optimal ones in all experiments in this thesis.

5 EXPERIMENTAL RESULTS

This thesis uses two MARL domains to evaluate the proposed EORP. In the first domain, called Coordinated Multi-agent Object Transportation Problem (CMOTP, presented in Section 5.1), two agents must coordinate their actions to solve a cooperative task. In such domain, the following characteristics of EORP are evaluated:

- the automatic selection of reward features (Section 5.1.4.1);
- the scalability of the resulting optimization process in a cooperative scenario (Section 5.1.4.2);
- the comparison of ORFs obtained by EORP against a reward function provided by the designers of the problem; and
- the application of EORP in single agent RL problems (a single-agent variant of CMOTP is provided in Section 5.1.4.3).

The second domain, called Traffic Assignment Problem (TAP, presented in Section 5.2), presents a MARL task in a non-cooperative setting, which involves thousands of selfish agents learning simultaneously. In this domain, the following characteristics of EORP are evaluated:

- The scalability of EORP in a non-cooperative scenario (Section 5.2.5.1);
- The ORFs obtained by EORP and compare them against a difference rewards technique and a deterministic method for the problem (Section 5.2.5.1);
- The use of different variable representation for reward features (Section 5.2.5.2);
- The application of EORP in different domain instances (Section 5.2.5.2); and
- The use of EORP in learning algorithms with different MDP representation for a same learning task (Section 2.3).

5.1 Coordinated Multi-agent Object Transportation Problem

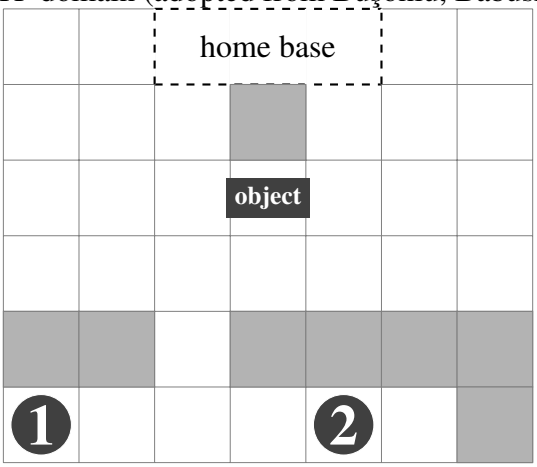
This section is organized as follows. The problem statement and learning scenario are presented in Section 5.1.1. The learning algorithm used to solve CMOTP is presented

in Section 5.1.2. Section 5.1.3 presents and discusses the basic setup of the learning algorithm. Finally, Section 5.1.4 discusses the obtained results.

5.1.1 Problem Statement and Scenario

The coordinated multi-agent object transportation problem (CMOTP), introduced in Buşoniu, Babuška and Schutter (2010), is an abstraction of a task involving the coordinated transportation of an object by two agents. The authors propose the CMOTP in the scenario represented in Figure 5.1, where two agents (circles) must travel on a grid 7×6 and transport the object (rectangle) to the home base (dashed line) as fast as possible. The task involves the avoidance of obstacles (grey cells) and coordinated moves.

Figure 5.1: The CMOTP domain (adopted from Buşoniu, Babuška and Schutter (2010)).



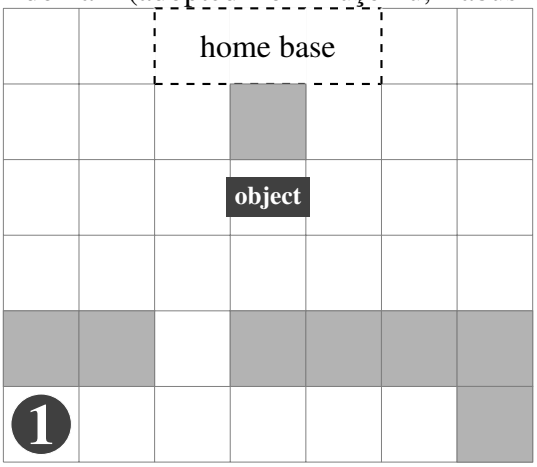
The agents start from the positions indicated in Figure 5.1. At each time step, they can move one cell to the left, right, up, down or stand still. The movements in the grid involve constraints: i) if an agent chooses to move to a cell that is not empty, it does not move; ii) if both agents try to go to the same cell, they do not move. Both agents are needed to move the object. The object only moves when both agents already grasped the object and pull it in the same direction. It can only be grasped from its left or right side. When an agent reaches a cell immediately to the left or the right of the object, it automatically grasps it. Once grasped, the object cannot be released. The task is accomplished when the object is taken to the home base.

This scenario presents two coordination issues. The first is to decide which agent will first pass through the narrow corridor. The second is to determine whether they should transport the target around the left or right side of the obstacle situated below the object.

A single-agent version of this problem, called single-agent object transportation

problem (SOTP), is also evaluated in this thesis. In this alternative problem, illustrated in Figure 5.2, agent 2 is removed from the scenario, and it is assumed that agent 1 has enough power to move the object alone.

Figure 5.2: The SOTP domain (adopted from Buşoniu, Babuška and Schutter (2010)).



5.1.2 Learning Algorithm

The agents learn their policies independently from each other (BUŞONIU; BABUSKA; SCHUTTER, 2008). The decision making process of the agents is modeled as a finite Markov Decision Process, composed of a set of states S and a set of actions A. For each state-action pair, $Q(s, a)$ represents the expected future reward that follows from executing action a in state s. The goal is to find a policy that maximizes the reward of the agent over its lifetime. In the CMOTP, state variables correspond to the coordinates $p_{i,X} \in \{1, 2, \ldots, 7\}$, $p_{i,Y} \in \{1, 2, \ldots, 6\}$ of each agent i, and a variable indicating whether a given agent i is currently grasping the object (and if so, from which side): $g_i \in \{\text{FREE, GRASPING-LEFT, GRASPING-RIGHT}\}$, for each agent $i \in I$. Therefore, each state $s \in S$ is a 6-dimensional vector $s = [p_{1,X}, p_{1,Y}, g_1, p_{2,X}, p_{2,Y}, g_2]$. At each state s, the agents can execute an action $a \in A = \{\text{LEFT, RIGHT, UP, DOWN, STAND-STILL}\}$. The complete state space has $|X| = (7 \times 6 \times 3)^2 = 15876$ elements. Some states are not valid because events such as collisions prevent certain combinations from occurring. The Q-values of each agent are updated via Q-learning (WATKINS; DAYAN, 1992), as in Equation 2.2.

The action selection is performed according to the ϵ - decreasing strategy $\epsilon_\lambda = \epsilon_0 \Delta\epsilon^\lambda$, where the exploration probability is initialized as ϵ_0 and exponentially decreases after each episode λ by a factor $\Delta\epsilon$. Through this strategy, agents choose actions randomly

(exploration) with probability ϵ, and greedily (exploration) with probability $1 - \epsilon$.

The reward function proposed by CMOTP's authors (Buşoniu, Babuška and Schutter (2010)) is presented in Equation 5.1. This function only rewards actions immediately related to the goal of the task; i.e., grasping the object and placing it at the home base. Agents do not have incentives or punishments for taking actions that are not directly related to the goal. It is worth mentioning that in Buşoniu, Babuška and Schutter (2010) the focus of the work is to demonstrate coordinated behavior in MARL and not the design of optimal reward functions (according to the EORP criterion), as in this thesis. Nevertheless, this function is used as a baseline reward function for comparison purposes in the experiments of this thesis.

$$R^B(s,a) = \begin{cases} 1, & \text{if the object is at the home base} \\ 0.1, & \text{if the agent grasped the object} \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

5.1.3 Basic Setup

For both SOTP and CMOTP, consider the following fitness and effort functions. The fitness function of each agent i is given by $f_1(h_i) = t_\lambda$, where t is the amount of time steps the agent spent to accomplish the task in episode λ. This function considers only the fitness produced in the last episode of the agent's lifetime. The effort function is given by $f_2(h_i) = \dfrac{\sum_{\lambda=0}^{\Lambda} t_\lambda}{|\Lambda|}$, which represents the average number of time steps spent during the lifetime of the agent. The reward design space used in SOTP is defined by $\mathcal{R}(J_S) \subset \mathcal{R}^{13}$, where the set of reward features J_S is defined in Table 5.1. For CMOTP domain, the reward design space is defined by $\mathcal{R}(J) \subset \mathcal{R}^{19}$, where J is the set of reward features defined in Table 5.2. These spaces capture the most common situations faced by the agents. In both $\mathcal{R}(J_S)$ and $\mathcal{R}(J)$ the reward feature called "otherwise" is always used, i.e., its corresponding indicator feature P is always activate. This feature represents any situations not directly represented in the search space. This is the reason for SOTP and CMOTP having, respectively 13 and 19 decision variables.

Learning policies in CMOTP requires setting some parameters: the learning rate (α), the discount factor (γ), the exploration rate (ϵ_0 and $\Delta\epsilon$), and the learning horizon (Λ). These parameters were experimentally set to: $\alpha = 0.8$, $\gamma = 0.99$, $\epsilon_0 = 1$, $\Delta\epsilon = 0.99$,

Table 5.1: Reward features of SOTP.

$\mathbf{J_S}$	Description
j_0	if the object is at the home base
j_1	if the agent grasped the object
j_2	if the agent hit a wall
j_3	if the agent cannot move the object alone
j_4	if the agent tried to go to a cell occupied by an object
j_5	if the agent chose to stand still
j_6	otherwise

Table 5.2: Reward features of CMOTP.

J	Description
j_0	if the object is at the home base
j_1	if the agent grasped the object
j_2	if the agent hit a wall
j_3	if the agent cannot move the object alone
j_4	if the agent tried to go to a cell occupied by an agent
j_5	if the agent tried to go to a cell occupied by an object
j_6	if other agent tried to go to a same cell
j_7	if the agent chose uncoordinated action to move the object
j_8	if the agent chose to stand still
j_9	otherwise

$\Lambda = 1000$. Learning policies that optimize the reward function in Equation 5.1 yields a baseline solution for CMOTP (R^B). The corresponding performance results are reported in Table 5.3. These results represent the average value and standard deviation of f_1 and f_2 over 30 runs. CMA-ES was used with population sizes of 20 elements throughout all experiments in this thesis.

5.1.4 Numerical Results

Section 5.1.4.1 discusses the reward features automatically selected in CMOTP. Section 5.1.4.2 demonstrates the scalability of EORP by comparing it with a strategy that optimizes one reward function per learning agent. Finally, in Section 5.1.4.3, it is demonstrated that EORP is also capable of dealing with single-agent problems.

5.1.4.1 Automatic Reward Feature Selection

The baseline reward function R^B (Equation 5.1) is composed of three features. However, many other features can compose the reward function. In Table 5.2, several situations that may be faced by CMOTP agents are presented. For the designer, it is not

easy to manually determine which features and respective reward signals best represent the learning task. The objective of the first experiment is to show that EORP helps the designer in this task.

The experiment consists in, given the set of potential features in Table 5.2, finding which features and reward signals produce at least one optimal reward function, R^*, that solves the task with maximum fitness and minimum effort. Table 5.3 presents the results obtained by solving the EORP and compare them with the performance resulting from using the baseline reward function. In the 30 repetitions, the EORP generated 102 Pareto-optimal ORFs from which R^* was selected for comparison purposes. From the ORFs that produced the best fitness value, R^* is the one that also produced the best effort among these. The column $\overline{R^*}$ in Table 5.3 represents the average fitness, average effort, and frequency with which each feature was used in the 102 Pareto-optimal solutions. The solution produced by the EORP, R^*, perform better than the baseline regarding both fitness and effort. The fitness produced by R^* is closer to the optimal solution for CMOTP, which is 12 steps. A reduction of $\approx 30\%$ in the learning effort is also observed when compared to the baseline reward function. Seven features were automatically selected by the solver to compose R^*. The features directly related to the success of the task, j_0 and j_1, are the only ones to receive positive reward signals. Furthermore, as shown in the $\overline{R^*}$ column of Table 5.3, these features were used by almost all computed solutions. The rest of the features utilized in R^* are associated with negative reward signals that punish those situations. The three unused features, j_3, j_6 and j_7, are also the least frequently used in $\overline{R^*}$.

This experiment shows that EORP can identify the features and reward signals that compose an ORF which, when optimized, produces maximum fitness and minimum effort. In simulated scenarios such as CMOTP, it is feasible to assume that it is acceptable to use as many reward features as necessary since there is a low computational cost associated with the use of each feature. However, in real scenarios, this does not hold because there may be other costs or constraints related to the use of each feature. For instance, how could a physical robot differentiate if it hit a wall, object or another robot? It could do so by having additional sensors embedded in its hardware, but doing so results in monetary, energetic and processing costs. Therefore, in certain situations, it may be worthwhile to use the minimum amount of features in an ORF.

The next experiment demonstrates that EORP can also find a solution that uses the minimum amount of features. To make this possible, a third objective function

Table 5.3: Fitness (f_1) and effort (f_2) yielded by the baseline (R^B) and solutions obtained by EORP. Column $\overline{R^*}$ represents the average fitness and effort for the solutions produced in all runs, as well as the frequency with which each feature was used. Column R^* represents the results obtained by the solutions with the lowest fitness and effort.

	$\mathbf{R^B}$	$\mathbf{R^*}$	$\overline{\mathbf{R^*}}$
$\mathbf{f_1}$	17.83 ± 1.74	12.4 ± 0.72	12.76 ± 0.85
$\mathbf{f_2}$	37.97 ± 3.03	26.83 ± 1.29	28.98 ± 2.54
j_0	1	0.77	97.1%
j_1	0.1	0.86	100%
j_2	-	-0.47	60.2%
j_3	-	-	28.2%
j_4	-	-0.08	57.3%
j_5	-	-0.28	49.5%
j_6	-	-	28.2%
j_7	-	-	47.6%
j_8	-	-0.59	71.8%
j_9	0	-0.23	100%

was added to the evaluation function $\mathcal{F}$. Consider the evaluation function $\mathcal{F}(H) = [f_1(H), f_2(H), f_3(H)]$, where $f_3(H) = \sum_{k=1}^{|J|} P(j_k)$ represents the amount of indicator features $P(j_k)$ used in a given potential solution. Note how the use of a more general formulation of the designer's objective, via a multi-objective evaluation function, instead of a single fitness function, allows the designer to easily specify multiple, possibly conflicting objectives that an optimal reward function is supposed to meet.

By considering this new evaluation function, the set of Pareto-optimal solutions produced in 30 runs is composed of 350 elements. Between these solutions (that take into account the trade-off between three goals: fitness, effort, and amount of activated features), this thesis is interested in the ones that use the minimum amount of features and that produce high fitness and low effort. Table 5.4 shows the performances of the three selected reward functions with such characteristics. The reward functions R_1^*, R_2^* and R_3^* use, respectively, one, two and three features and are in the Pareto-optimal set. All three functions performed better in the CMOTP domain than the baseline R^B, which uses features j_1, j_2 and j_9. A t-test conducted with 95% of confidence interval shows that, regarding fitness, the three solutions are equivalent. Regarding effort, only reward function R_1^* does not overcome the baseline.

The reward function R_1^* uses only one feature, j_9, which is mandatory by definition. The reward signal associated with this feature by optimizing the EORP criterion punishes the agent at each step. Agents learning with this function thus attempt to minimize the punishment received during their lifetime. As there is no direct incentive (posi-

tive reward) for reaching the goal state, the agent takes longer to learn its behavior. In the reward function R_2^*, a second feature, j_1, which rewards the goal state positively, is used. The use of j_1 reduces the learning effort by $\approx 55\%$ when compared with R_1^*. The same occurs with R_3^*, which uses a third feature, j_8, to punish the agent when it chooses the action stand-still, resulting in a reduction of learning effort of 9% compared to R_2^*.

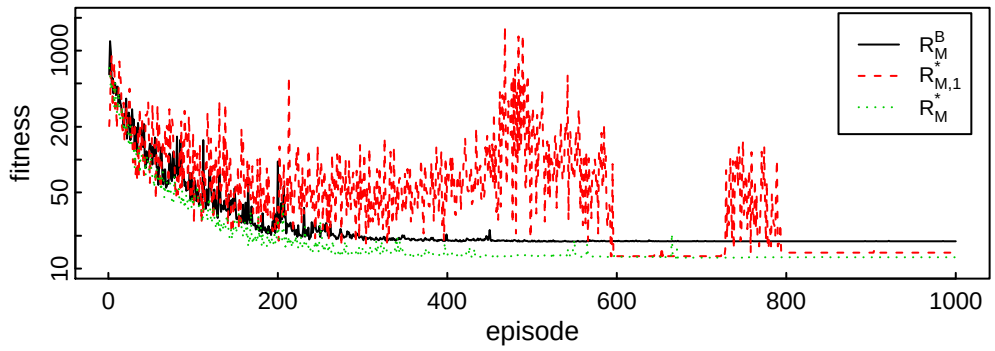
Table 5.4: Fitness (f_1), effort (f_2) and number of features (f_3) obtained by solutions that use one (R_1^*), two (R_2^*), and three (R_3^*) features.

	$\mathbf{R_1^*}$	$\mathbf{R_2^*}$	$\mathbf{R_3^*}$
$\mathbf{f_1}$	13.68 ± 0.96	13.78 ± 0.98	13.76 ± 1.01
$\mathbf{f_2}$	77.54 ± 4.15	34.89 ± 2.21	31.67 ± 1.86
f_3	1	2	3
j_1	-	0.73	0.66
j_8	-	-	-0.54
j_9	-0.18	-0.1	-0.08

To show the effects of a high learning effort, Figure 5.3 presents the convergence curve for the solutions with the highest (R_1^*) and lowest (R^*; see Table 5.3) learning effort obtained by EORP, as well as the one yielded by the baseline (R^B). Error bars are omitted to make the graph less cluttered, but variance can be inferred by analyzing the amplitude with which the curves vary. The reward function R^* results in faster convergence. This function uses 7 features to reward the agent. It makes it able to identify state-action pairs that must be avoided. Thus, even with a high exploration rate in early episodes, the agent learns its behavior faster. The function R_1^* results in slow convergence due to its use of a relatively poorer/less informative set of reward features. For instance, if an agent learning with R^* hits a wall, it receives an immediate punishment that differentiates this action from others. If the same situation occurs with an agent learning under R_1^*, the reward signal received is the same that it would receive by having performed any other action. So, immediately it does not know whether it is appropriate to hit a wall. This is why even though it yields better fitness than the baseline, the convergence of R_1^* is very slow. R^B results in smoother convergence than others because it only rewards states directly related to the goal. Any action that leads the agent to non-goal related states has a neutral reward (zero). This avoids the propagation of noise generated by random actions to the rest of the MDP but makes it difficult for agents to learn appropriate behaviors. This is why the baseline reward function produces worse fitness than the reward functions identified by optimizing EORP.

This experiment shows that the quality of the learned behavior is directly related to the features and reward signals used in the constructed reward functions. The adequate use

Figure 5.3: Performance (fitness) vs. time (episode) in CMOTP.



of features and reward signals improves both fitness and effort associated with learning a task. Although reward functions with fewer features do converge to desirable behavior, convergence speed is negatively impacted. It is up to the system's designer to determine the feasibility of using solutions with more features. In real word problems, for instance, even with energy and monetary costs associated with the use of additional features, the benefits gained regarding convergence time can make those costs acceptable.

5.1.4.2 Scalability in Multi-Agent Settings

This experiment demonstrates that EORP is scalable in the presence of multiple learning agents. As shown before, the EORP optimizes a single reward function $R^* \in \mathcal{R}(J)$ per collective of agents I, resulting in a search space dimensionality given by $D(\mathcal{R}(J)) = 2|J|$. The existing approaches for dealing with ORP in multi-agent settings optimize one reward function, $R^*_i \in R^*_I$ per agent $i \in I$, such that $R^*_I \subset \mathcal{R}$. The dimensionality of the resulting ORP is given by $D(\mathcal{R}(J), I) = (2|J|) \times |I|$. Note that the number of decision variables increases according to the number of learning agents. In the search space of CMOTP, for instance, presented in Table 5.2, the optimization problem is composed by $D(\mathcal{R}_M) = 9 + 10 = 19$ decision variables, whilst in approaches such as Liu et al. (2014), which optimize one reward function per agent, the search space is composed by $D(\mathcal{R}_M, I) = (9 + 10) \times 2 = 38$ decision variables.

In Table 5.5, the performance of both strategies is compared. The results presented for the strategy that uses a single reward function, R^* for all agents were discussed before. The strategy that uses one reward function per agent, R^*_I (where $R^*_{i=1}$ and $R^*_{i=2}$ are, respectively, the reward functions of agent 1 and 2) yield worse performance when compared to R^*. A t-test with 95% of confidence interval was conducted and showed that R^* overcomes R^*_I in both fitness and effort. The learning effort presented by R^*_I is $\approx 221\%$ higher than when using R^*. The reward functions $R^*_{i=1}$ and $R^*_{i=2}$ use different

Table 5.5: Fitness (f_1) and effort (f_2) produced by systems that use a single (R^*) or multiple (R^*_I) reward functions when solving CMOTP. Columns $R^*_{i=1}$ and $R^*_{i=2}$ represent, respectively, the reward functions of agent 1 and 2.

	$\mathbf{R^*}$	$\mathbf{R^*_I}$			
		$\mathbf{R^*_{i=1}}$	$\mathbf{R^*_{i=2}}$	$\overline{\mathbf{R^*_{i=1}}}$	$\overline{\mathbf{R^*_{i=2}}}$
$\mathbf{f_1}$	12.4 ± 0.72	13.47 ± 1.01		13.79 ± 1.02	
$\mathbf{f_2}$	26.83 ± 1.29	59.33 ± 4.53		57.97 ± 11.98	
j_0	0.77	0.79	0	85.7%	89.3%
j_1	0.86	-	0	42.9%	67.9%
j_2	-0.47	-0.68	-1	28.6%	39.3%
j_3	-	-	-	53.6%	42.9%
j_4	-0.08	-	-	32.1%	46.4%
j_5	-0.28	-0.84	-1	28.6%	42.9%
j_6	-	-0.23	-	60.7%	50%
j_7	-	-	-1	39.3%	64.29%
j_8	-0.59	-0.79	-	75%	50%
j_9	-0.23	-0.21	-1	100%	100%

sets of features and reward signals for each agent. From the 6 features used in $R^*_{i=1}$, only one rewards positively agent 1 (j_0) in a goal related feature. All others are used to punish it in non-goal related features. The reward function $R^*_{i=2}$ associates neutral reward signals with the two goal-related features, and 4 features to punish agent 2 with negative rewards. Columns $\overline{R^*_{i=1}}$ and $\overline{R^*_{i=2}}$ represent the average fitness and effort for the solutions produced, as well as the frequency with which each feature was used. Except for feature j_9 (which is mandatory), only feature j_0 was used by all computed solutions. The frequency of use of different reward feature indicates that reward functions obtained by R^*_I, in general, use a different set of features for each agent. Instead of improving the fitness and effort, this strategy results in worse overall performance.

5.1.4.3 EORP in Single-Agent Settings

In the experiments shown so far, EORP was applied to multi-agent settings. To demonstrate that the optimization criterion is general enough for dealing with both multi and single-agent settings, the present experiment applies EORP to the SOTP. It is considered the reward design space composed by the reward features presented in Equation 5.1 and the evaluation function introduced in Equation 4.3.

Table 5.6 presents the results obtained by EORP and compare them against the baseline reward function (Equation 5.1). Column $\overline{R^*_S}$ presents the average results of all solutions produced over 30 runs. The solution identified by EORP can achieve the optimal fitness for the problem (8 steps) in all cases. This is the reason why the standard deviation

is zero in line f_1. The same occurs for the baseline reward function. A t-test with 95% of confidence was conducted to compare the effort resulting from the use of R_S^* and R^B. R_S^* overcomes R^B regarding effort, and R_S^* uses all features available in the reward design space. The two features directly related to the goal of the task (j_0 and j_1) are the only ones to be rewarded positively in R_S^*. Column $\overline{R_S^*}$ shows that these features are also used by all solutions produced over 30 runs. All other features used in R_S^* are associated with punishments to the agent. This is different from R^B, which provides the agent with a zero reward in all situations except the ones represented by the features j_0 and j_1.

Table 5.6: Fitness (f_1) and effort (f_2) produced by the baseline (R^B) and ORF (R_S^*) in SOTP. The results are averages over 30 runs. Column $\overline{R_S^*}$ represents the average fitness and effort of the solution produced over all runs, as well as the frequency with which each feature is used. Column R_S^* represents the results obtained by the solutions with lowest fitness and effort.

	$\mathbf{R^B}$	$\mathbf{R_S^*}$	$\overline{\mathbf{R_S^*}}$
$\mathbf{f_1}$	8.0 ± 0.0	8.0 ± 0.0	8.0 ± 0.0
$\mathbf{f_2}$	11.1 ± 0.35	10.87 ± 0.4	10.71 ± 0.04
j_0	1	0.86	100%
j_1	0.1	0.73	100%
j_2	-	-0.32	48%
j_3	-	-0.44	44%
j_4	-	-0.63	44%
j_5	-	-0.74	64%
j_6	0	-0.31	100%

The experiments conducted above, designed to evaluate EORP in single-agent scenarios, indicate that the optimized reward functions are similar in performance to what is achieved by the baseline reward function, which was manually designed by an expert. This suggests that R^B captures well the relevant reward features for solving the SOTP task efficiently both regarding fitness and effort. Note that although the baseline (provided *a priori* by a designer) performed well, EORP was capable of *autonomously* identifying equally effective reward functions, which suggests that it does, indeed, apply not only to multi-agent settings but also to single-agent cases where hard-coded solutions expertly constructed by a designer are available.

5.2 Traffic Assignment Problem

This section is organized as follows. Section 5.2.1 presents the problem statement. The three scenarios used in the experiments are presented and discussed in Section 5.2.2.

Section 5.2.3 presents the two learning algorithms used for solving the TAP. The basic settings of each algorithm are represented in Section 5.2.4. Experimental results are discussed in Section 5.2.5.

5.2.1 Problem Statement

The traffic assignment is an important stage in the task of modeling and simulating a transportation system. A transportation system can be represented as a composed of two parts: supply and demand. Traffic assignment methods connect the physical infrastructure (supply) to road users (demand) that are going to use it, i.e., it assigns trips to each edge of the road network (ORTÚZAR; WILLUMSEN, 2011). This thesis addresses the traffic assignment problem (TAP) in a decentralized perspective, in which road users (agents) must learn individually the best route that satisfies their origin and destination constraints.

The supply part of a transportation system can be modeled as a graph $G\left(V, E\right)$, where:

- V is a set of vertices that represent network intersections;
- E is a set of edges (or links) between these vertices, which represent the road sections; and
- c_e of an edge $e \in E$ represents a form of travel cost associated with the crossing of the edge e.

The travel cost of an edge may be represented by several metrics: travel time, fuel spent and travel distance are good examples of measures used in the literature. These metrics are usually abstracted by functions known as volume-delay functions (VDF). A good example of VDF, which is widely used in the literature, is the one proposed by Bureau (1964):

$$c_e = t_e^0 \left[1 + \mathcal{A}\left(\frac{\mathcal{V}_e}{C_e}\right)^{\mathcal{B}}\right] \tag{5.2}$$

where:

- c_e represents the travel cost in terms of time for crossing edge e;
- t_e^0 is the travel time per unit of time under free-flow conditions (free-flow travel time) of edge e;

- $\mathcal{V}_e$ is the flow of vehicles (vehicles per unit of time) using edge e;
- C_e is the edge capacity; and
- $\mathcal{A}$ and $\mathcal{B}$ are parameters specifically defined for each edge, related to its physical characteristics.

The second part of the transportation system, the demand, represents the users of the infrastructure. The demand can be represented by an origin-destination matrix (OD-matrix). An OD-matrix T contains I lines (origin zones) and J columns (destination zones). Each element T_{ij} represents the number of trips from vertex i to vertex j in a given time interval. It is said that $i \in I$ and $j \in J$ form an OD-pair.

Solving the TAP consists of connecting the supply and demand by assigning routes to each road user. Each route consists of a set of edges, forming a route between an origin and a destination vertex. Thus, a route p is defined by a sequence of connected vertices $(v_0, v_1, v_2, \ldots)$. The route cost function ($c(p)$) defined in Equation 5.3 represents the sum of travel time costs c_e of all edges $e \in E_p$ for a given route p, where E_p is the set of connected edges that composes p.

$$c(p) = \sum_{e \in E_p} c_e \tag{5.3}$$

The user equilibrium (UE) is a solution concept for the TAP proposed by Wardrop (1952). It is commonly used to evaluate the quality of an assignment. An UE assignment is reached when no road user may lower their travel cost through unilateral action. As a performance measure, it is considered the average travel time (ATT), as in Equation 5.4:

$$\text{ATT} = \sum_{i \in I} \frac{c(p_i)}{|I|} \tag{5.4}$$

This measure represents the average travel times (in minutes) of the routes used by the agents. The present thesis uses the ATT under UE condition as a baseline solution of a TAP.

This section is organized as follows. Section 5.2.1 presents the problem statement. The three scenarios used in the experiments are presented and discussed in Section 5.2.2. Section 5.2.3 presents the two learning algorithms used for solving the TAP. The basic settings of each algorithm is represented in Section 5.2.4. Experimental results are discussed in Section 5.2.5.

5.2.2 Scenarios

Three TAP scenarios that differ regarding network topology and number of agents are used. The first scenario, called OW, was proposed in Ortúzar and Willumsen (2011, Chapter 10). The network topology of this scenario is illustrated in Figure 5.4, where the values over the edges are their free-flow travel times. The road network contains 13 vertices and 24 two-way edges. The demand is composed by a constant flow of 1700 trips distributed over the 4 OD-pair presented in Table 5.7. The OW cost function is given as in Equation 5.5:

$$c_e = t_e^0 + V_e \times a \tag{5.5}$$

where:

- c_e is the travel time in minutes to cross edge e;
- t_e^0 is the free-flow travel time for edge e;
- V_e is the flow using e; and
- a is the increment that linearly increases the travel time in a minutes per vehicle using e.[1]

Figure 5.4: Network topology of scenario OW.

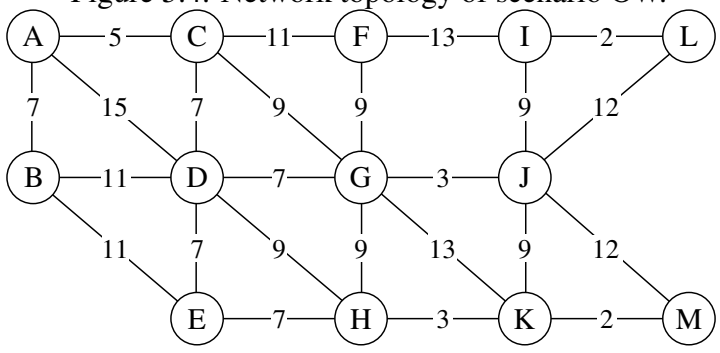


Table 5.7: OD-matrix of scenario OW.

Origin	Destination	Trips
A	L	600
A	M	400
B	L	300
B	M	400

[1] OW scenario uses a fixed increment of 0.02 minutes in all of its edges.

The second TAP scenario is an adaptation of a scenario presented by Nguyen and Dupuis (1984), called scenario ND. The original road network was modified so that all roads are two-way to provide more options for routes. The resulting graph, represented in Figure 5.5, has 13 vertices and 38 edges. The demand, presented in Table 5.8, is composed of 2000 trips distributed over 4 OD-pairs. The VDF of the ND scenario is defined as in Equation 5.5. The complete description of the values of the parameters of each link (parameters t_e^0 and a) are available in Nguyen and Dupuis (1984).

Figure 5.5: Network topology of scenario ND.

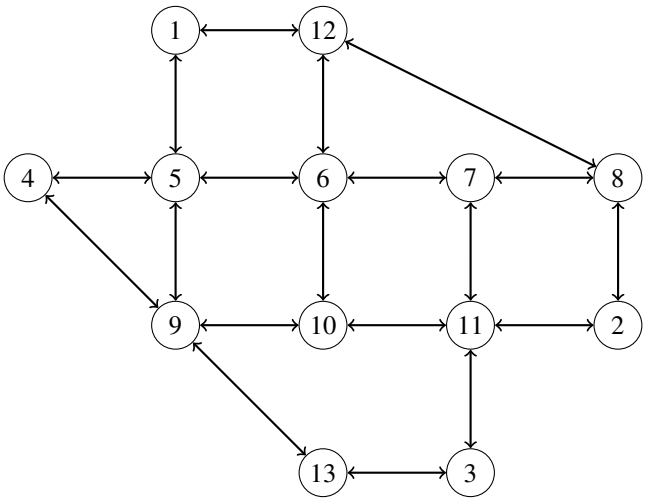


Table 5.8: OD-matrix of scenario ND.

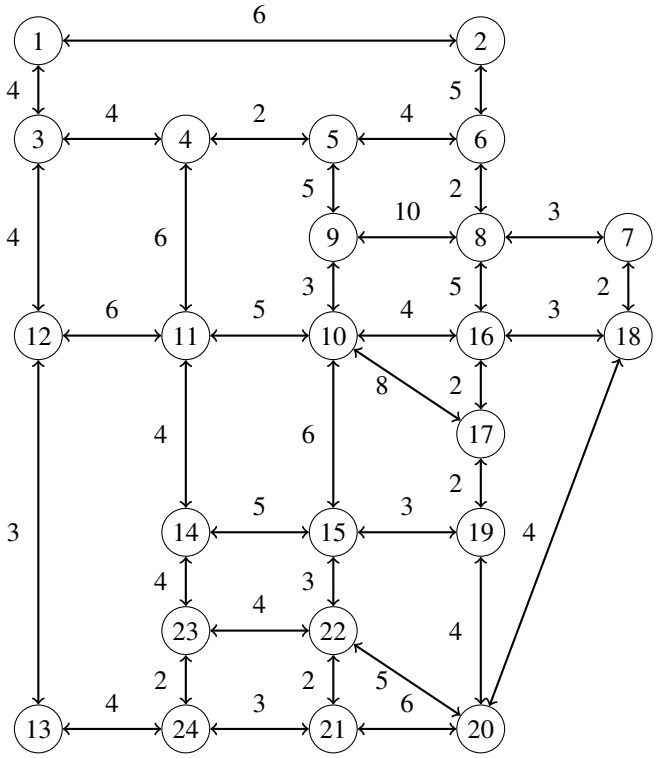
Origin	Destination	Trips
1	2	400
1	3	800
4	2	600
4	3	200

The third and larger TAP scenario used in this thesis is the Sioux Falls (SF) network. This is a well-known transportation problem used in the literature as a testbed for traffic assignment methods[2]. The road network, shown in Figure 5.6, has 24 vertices and 76 edges, where the values over the edges are their free-flow travel times. The demand, presented in Table 5.9, is comprised of 360600 trips distributed among 528 OD-pairs. The cost function of scenario SF is defined as in the VDF of Equation 5.2, where the values of parameters $\mathcal{A}$ and $\mathcal{B}$ are respectively defined in 0.15 and 4 for every link of this network.

Relevant aspects of these three scenarios are summarized in Table 5.10. This table also presents for each scenario, its ATT under UE condition. The ATTs under UE condition are obtained through the method of successive averages (ORTÚZAR; WILLUMSEN,

[2]All data sets containing network, demand, cost function and solutions of scenario SF are available at <http://github.com/bstabler/TransportationNetworks>.

Figure 5.6: Network topology of scenario SF.



2011, Chapter 10) (MSA). This method is a well-known centralized and deterministic method for solving the TAP that provides convergence guarantees to an approximated UE solution. Note that, from the point of view of MARL, the SF network seems to present more challenging issues than others, since it corresponds to a larger network topology, and has demand up to 200 times bigger and better distributed over its OD-pairs. This large demand for scenario SF provides a highly competitive environment for the agents, which must compete for the most attractive edges.

5.2.3 Learning Algorithms

The present thesis uses two MARL approaches that were previously discussed in the work of Grunitzki and Bazzan (2017) for solving the TAP. These approaches are called edge-based-QL and route-based-QL. They both model the agent's decision-making process as an MDP. The main difference between these two approaches is that the edge-based-QL does not restrict the agents' search space. Rather, it enables agents to experience all possible routes available in the road network. Consequently, it makes the learning task more complex due to the larger search space it provides. On the other hand, the route-based-QL restricts the search space of the agents to a subset of precomputed routes, which requires prior knowledge about the road network to compute such routes.

Table 5.9: OD-matrix of scenario SF. The rows are the origin and the columns are the destinations. The number in the cells represent the amount of trips in thousands.

OD	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	1	1	5	2	3	5	8	5	13	5	2	5	3	5	5	4	1	3	3	1	4	3	1
2	1	0	1	2	1	4	2	4	2	6	2	1	3	1	1	4	2	0	1	1	0	1	0	0
3	1	1	0	2	1	3	1	2	1	3	3	2	1	1	1	2	1	0	0	0	0	1	1	0
4	5	2	2	0	5	4	4	7	7	12	14	6	6	5	5	8	5	1	2	3	2	4	5	2
5	2	1	1	5	0	2	2	5	8	10	5	2	2	1	2	5	2	0	1	1	1	2	1	0
6	3	4	3	4	2	0	4	8	4	8	4	2	2	1	2	9	5	1	2	3	1	2	1	1
7	5	2	1	4	2	4	0	10	6	19	5	7	4	2	5	14	10	2	4	5	2	5	2	1
8	8	4	2	7	5	8	10	0	8	16	8	6	6	4	6	22	14	3	7	9	4	5	3	2
9	5	2	1	7	8	4	6	8	0	28	14	6	6	6	9	14	9	2	4	6	3	7	5	2
10	13	6	3	12	10	8	19	16	28	0	40	20	19	21	40	44	39	7	18	25	12	26	18	8
11	5	2	3	15	5	4	5	8	14	39	0	14	10	16	14	14	10	1	4	6	4	11	13	6
12	2	1	2	6	2	2	7	6	6	20	14	0	13	7	7	7	6	2	3	4	3	7	7	5
13	5	3	1	6	2	2	4	6	6	19	10	13	0	6	7	6	5	1	3	6	6	13	8	8
14	3	1	1	5	1	1	2	4	6	21	16	7	6	0	13	7	7	1	3	5	4	12	11	4
15	5	1	1	5	2	2	5	6	10	40	14	7	7	13	0	12	15	2	8	11	8	26	10	4
16	5	4	2	8	5	9	14	22	14	44	14	7	6	7	12	0	28	5	13	16	6	12	5	3
17	4	2	1	5	2	5	10	14	9	39	10	6	5	7	15	28	0	6	17	17	6	17	6	3
18	1	0	0	1	0	1	2	3	2	7	2	2	1	1	2	5	6	0	3	4	1	3	1	0
19	3	1	0	2	1	2	4	7	4	18	4	3	3	3	8	13	17	3	0	12	4	12	3	1
20	3	1	0	3	1	3	5	9	6	25	6	5	6	5	11	16	17	4	12	0	12	24	7	4
21	1	0	0	2	1	1	2	4	3	12	4	3	6	4	8	6	6	1	4	12	0	18	7	5
22	4	1	1	4	2	2	5	5	7	26	11	7	13	12	26	12	17	3	12	24	18	0	21	11
23	3	0	1	5	1	1	2	3	5	18	13	7	8	11	10	5	6	1	3	7	7	21	0	7
24	1	0	0	2	0	1	1	2	2	8	6	5	7	4	4	3	3	0	1	4	5	11	7	0

This thesis considers each learning agent as a road user associated with a trip in an OD-matrix. The origin node in the OD-matrix represents the initial state (s_-) of the agent, i.e., the state where the agent is created at the beginning of each learning episode. The destination node in the OD-matrix represents the agent's terminal state (s_+), i.e., the goal state of the agent. Each agent can learn a route between its origin and destination. For both edge-based-QL and route-based-QL, the learning process is organized in episodes (trials) and time steps (time unit). An episode λ ends in $t_{\max}$ time steps or when all agents reach their terminal state (whichever comes first).

Following the standard practice in the literature (which abstracts the network as a graph), each unit of time step $t \leq t_{\max}$ represents a hop in the graph. For instance, the route $p = (v_0, v_1, v_2)$ that connects v_0 to v_2 requires $t = (|p| - 1)$ time steps to be traveled.

The following sections present the details of edge-based-QL (Section 5.2.3.1) and route-based-QL (Section 5.2.3.2), which will be used as RL algorithms in the TAP exper-

Table 5.10: Relevant aspects of scenarios OW, ND and SF.

Feature	OW scenario	ND scenario	SF scenario
Trips	1700	2000	360600
OD-pairs	4	4	528
Vertices	13	13	24
Edges	24	38	76
ATT UE	≈ 67.16	≈ 50.28	≈ 20.78

iments of this thesis

5.2.3.1 Edge-based Q-Learning

This approach is called "edge-based" because it assumes that agent's actions correspond to edges of a road network. When an agent is learning with this method, it can find a policy from the set of all routes available in the search space. The agent's MDP is modeled as follows. A state s is a vertex $v \in V$ of the road network in which the agent is located. Each state $s \in S$ has a set of available actions $\mathcal{A}(s)$, which is represented by the outgoing edges of the corresponding vertex of s. The reward received by the agent is given by a reward function $R(s, a)$ that rewards the agent after performing an action.

If it is allowed for agents to drive in loops, this approach may present infinity routes $p \in \mathcal{P}$ from an initial state s_- to a terminal state s_+. To handle this, the agent's search space is limited by a maximum number of time steps $t_{\max}$. Thus, if the agents do not find a route from s_- to s_+ in $t_{\max}$ time steps, the current episode is stopped. Even though the stopped route belongs to an invalid set of routes $\mathcal{P}_{\text{invalid}}$, it is also a possible solution experienced by the agent. In this manner, the set of all possible routes for an edge-based-QL agent is defined as in Equation 5.6:

$$\mathcal{P}^{(s_-, s_+, t_{\max})} = \mathcal{P}_{\text{valid}} \cup \mathcal{P}_{\text{invalid}} \tag{5.6}$$

where:

- $\mathcal{P}_{\text{valid}} = \left\{ p_{i,j}^{(s_-, s_+)}(t_i) \right\} \mid i = 1, \ldots, M; j = 1, \ldots, N_i$, where M is the current simulation time step, and N_i is the number of routes from s_- to s_+ in t_i steps; and
- $\mathcal{P}_{\text{invalid}} = \left\{ p_{i,j}^{(S, V_{t_i})}(t_i') \right\} \mid i = 1, \ldots, M'; j = 1, \ldots, N_i'$, where M' is the current simulation time step, and N_i' is the number of routes from s_i to $v_{t_i'}$ in t_i' steps.

An important property of $\mathcal{P}$ for this approach is that: $\mathcal{P}_{\text{valid}} \cap \mathcal{P}_{\text{invalid}} = \emptyset$.

5.2.3.2 Route-based Q-Learning

Different from edge-based-QL, in route-based-QL the agent's action represents a route connecting its origin to its destination. The search space $\mathcal{P}$ is restricted to a predefined subset of possible solutions, as shown by Equation 5.7:

$$\mathcal{P}^{(s_-,s_+)} = \{p_1, \ldots, p_j\} \tag{5.7}$$

where j is the index of the j-th lowest cost route. This subset of routes is preprocessed before the learning process begins. Thus, each agent receives $|\mathcal{P}|$ precomputed routes from s_- to s_+.

The set of actions is defined according to the number of predefined routes used. A parameter $K = |\mathcal{P}|$ must be defined to determine the number of routes to be calculated. These routes are the K-lowest cost routes of the agent's OD-pair under no congestion. These routes are computed in the graph using the algorithm K Shortest Loopless Paths (YEN, 1971), which can find the K shortest routes without loops for an OD-pair.

The agent's MDP is modeled as follows. The state set of the agent contains only the initial and terminal states. At its initial state, the agent has $\mathcal{A}(s_-) = |\mathcal{P}|$ actions. Each action is a route that connects s_- to s_+. When the agent reaches the terminal state, it is rewarded.

In this approach, an episode finishes when all agents reach a terminal state. The $t_{\max}$ parameter is disregarded here because by construction no route has loops. This approach presents a major advantage when compared against edge-based-QL, which is the extra parameter ($|\mathcal{P}|$). Furthermore, even though the search space restriction can simplify the learning process, it may underestimate the ability of the agent to learn the most appropriated policies if the search space (i.e., the set of k shortest routes used to form the action set $\mathcal{P}$) was set incorrectly.

5.2.4 Basic Setup

This section presents the basic setup of EORP (Section 5.2.4.1), baseline reward functions (Section 5.2.4.2) and learning algorithms (Section 5.2.4.3) that are common to all TAP experiments. Specific settings of each experiment will be presented along the experiments (Sections 5.2.5.1 and 5.2.5.2).

5.2.4.1 EORP Settings

The multi-objective evaluation function is given as in Equation 5.8:

$$\mathcal{F}(H) = [-f_1(H), -f_2(H)] \tag{5.8}$$

where the fitness function (f_1) and learning effort function (f_2) are defined as follows. The fitness function of each agent i is given by $g_1(h_i) = c(p_i)$, which represents the negative cost of of the route p_i, learned by the agent i. This function considers only the fitness produced in a given (the current) episode. By using this function, the fitness is maximized when the travel time of the agent is minimized. The average fitness produced by all agents (f_1) is given as in Equation 4.4 (Page 38).

The effort function of an agent is given by $g_2(h_i) = \frac{\sum_{\lambda \in \Lambda} t_\lambda}{|\Lambda|}$, which represents the average number of decisions taken by the agent i during its lifetime. Consider the average effort produced by all agents (f_2), as in Equation 4.5 (Page 38).

The CMA-ES algorithm was used and empirically defined with a population size of 20 elements and stopping criteria of 1000 evaluations.

5.2.4.2 Baseline Reward Functions

Along with TAP experiments, two baseline strategies for reward function design are used for comparison purposes. The first strategy, called expert-designed (ED), represents the most direct strategy adopted by a MARL system's designer. This thesis uses the ED reward functions presented in Grunitzki and Bazzan (2017), which are also similar to the ones presented in other works that apply MARL algorithms for solving the TAP (GRUNITZKI; RAMOS; BAZZAN, 2014a; GRUNITZKI; BAZZAN, 2016; BAZZAN; GRUNITZKI, 2016; RAMOS; GRUNITZKI, 2015; RAMOS; GRUNITZKI; BAZZAN, 2014). In this strategy, the travel time resulting by an agent's action is used as a reward signal. For edge-based-QL, the ED reward function is given by Equation 5.9:

$$R_{ED} = -c_e \tag{5.9}$$

By learning with this reward function, at each action performed, the agent is rewarded by the negative travel cost (c_e) of the traveled edge corresponding to agent's action.

The expert-designed reward function for route-based-QL is defined as in Equa-

tion 5.10:

$$R_{ED} = \sum_{e \in E_p} -c_e \tag{5.10}$$

where E_p is the set of edges that composes route p. This is similar to the reward function in Equation 5.9. The difference is that in route-based-QL the feedback received by an agent's action refers to the negative travel cost of a route and not of a single edge.

The second baseline reward function design strategy is the difference rewards. For both edge-based and route-based-QL, the DR function in given as in Equation 5.11:

$$R_{DR} \equiv G(z) - G(z_{-i} + c_i) \tag{5.11}$$

where:

- $G(z) = \frac{\sum_{i \in I} c(p_i)}{|I|}$ represents the average travel time of the traveled routes of all agent in I; and
- $G(z_{-i})$ represents the average travel time of the routes of all agent in $I \setminus i$, i.e., the travel time of agent i is disregarded from the average travel time in $G(z_{-i})$.

5.2.4.3 Q-Learning Settings

The Q-learning algorithm and exploration strategy adopted in the learning methods has four parameters to be set: learning rate (α), discount factor (γ), number of episodes (Λ) and exploration rate (ϵ). The number of episodes was empirically set as 1000. The exploration policy utilized is the ϵ-decreasing, with $\epsilon = 1.0$ and $\Delta\epsilon = 0.99$. This basic setup is used by both algorithms.

The rest of the parameters (α, γ and $|\mathcal{P}|$) are defined on a case basis. In both learning algorithms, the performance measure considered is the ATT obtained by the expert-designed reward functions. All the reported experiments were repeated 30 times.

For discovering the best combination of α and γ for the edge-based-QL, different combinations of values in the interval $\alpha, \gamma \in [0, 1]$ were tested. The $t_{\max}$ parameter was defined as 100 since the optimal travel times are known (from the literature). The results obtained regarding ATT for OW scenario are presented in Table 5.11. The performance of edge-based-QL is more sensitive to γ than to α. This can be explained by the fact that actions (edges that depart from a given state) that can be taken at states (nodes) are very important in this problem since the agent is trying to make a series of decisions in order

to minimize the travel time in the whole route. Therefore the discount rate must be high. In the remaining of this paper, experiments regarding the edge-based-QL use $\alpha = 0.5$ and $\gamma = 0.99$. The same experiment was conducted for edge-based-QL in scenarios SF and ND. The algorithm presented similar behavior regarding its values of parameters.

Table 5.11: Average travel time of OW scenario for edge-based-QL (standard deviation in parentheses).

		α				
		0.1	**0.3**	**0.5**	**0.7**	**0.9**
γ	**0.1**	451.291 (11.114)	342.678 (7.726)	331.193 (10.156)	333.187 (7.491)	338.804 (7.184)
	0.3	282.363 (5.499)	219.678 (3.847)	209.316 (3.841)	205.834 (4.028)	204.814 (3.721)
	0.3	142.116 (2.506)	122.841 (1.372)	118.985 (1.044)	116.592 (1.625)	116.155 (2.037)
	0.7	81.277 (1.542)	76.089 (1.175)	75.352 (0.845)	75.204 (0.681)	75.475 (0.814)
	0.9	67.754 (0.223)	67.314 (0.022)	67.316 (0.026)	67.335 (0.041)	67.407 (0.067)
	0.99	67.281 (0.142)	67.154 (0.024)	**67.153 (0.018)**	67.152 (0.043)	67.155 (0.026)

The route-based-QL has two parameters to be set: α and $|\mathcal{P}|$ (number of pre-computed routes). In order to find the best combination for them, experiments in the OW scenario were performed evaluating the performance of the algorithm regarding ATT for the intervals of values of α and $|\mathcal{P}|$ presented in Table 5.12. The learning rate seems to play a minor role. On the other hand, the amount of pre-computed routes has a noticeable influence. Low values of $|\mathcal{P}|$ cause agents to compete for few routes. This allocates too much traffic flow in some edges, whilst others areas of the road network are underutilized. As the number of pre-computed routes increases, the ATT decreases. Student's t-tests were applied to these data and showed that for $|\mathcal{P}| = 8$ and $|\mathcal{P}| = 10$ the ATTs are equivalent. For the OW network, the combination of values of parameters was defined as $\alpha = 0.3$ and $|\mathcal{P}| = 10$. Similar behavior was observed in the ND scenario. In the SF scenario, the best combination was $\alpha = 0.9$ and $|\mathcal{P}| = 10$. The rest of this thesis uses these values of parameters for route-based-QL along all experiments

Table 5.12: Average travel time (ATT) and standard deviation (in parentheses) for route-based-QL in OW scenario.

		α				
		0.1	**0.3**	**0.5**	**0.7**	**0.9**
$\mathcal{P}$	**2**	83.937 (0.002)	83.952 (0.086)	83.933 (0.011)	83.993 (0.222)	83.954 (0.065)
	4	71.763 (0.005)	71.525 (0.181)	71.574 (0.138)	71.922 (0.462)	72.039 (0.440)
	6	67.370 (0.009)	67.302 (0.006)	67.336 (0.088)	67.354 (0.058)	67.721 (0.417)
	8	67.143 (0.005)	67.159 (0.035)	67.205 (0.0148)	67.267 (0.176)	67.505 (0.353)
	10	67.151 (0.007)	**67.162 (0.008)**	67.241 (0.190)	67.286 (0.236)	67.577 (0.377)

5.2.5 Numerical Results

The TAP experiments extend the analysis provided in Section 5.1.4.2 to show that EORP also scales to multi-agent problems that include thousands of learning agents. In the experiments presented in Section 5.2.5.1, an extensive comparison of EORP against DR was provided. Such analysis uses the edge-based-QL learning with EORP reward functions that only consider reward features that assume binary values (activated or deactivated). A more general performance comparison of edge-based-QL and route-based-QL under the guidance of reward functions provided by ED, DR and EORP reward functions is presented in Section 5.2.5.2. This latter experiment considers that EORP reward features may be represented by both binary or real values.

5.2.5.1 EORP versus Difference Rewards

The experiment presented in this section also appears in Grunitzki, Silva and Bazzan (2018). This experiment evaluates the performance edge-based-QL in solving the TAP under the guidance of reward functions provided by difference rewards and EORP.

First, the experiment-specific settings are presented. For EORP, consider the reward design space defined by $\mathcal{R}(J) \subset \mathcal{R}^7$, where J is the set of reward features defined in Table 5.13. This space captures some of the most common situations faced by the agents in the TAP. Note that it is just using reward features that have binary representation, i.e., reward features that just assume the values "activated" or "deactivated". The feature called "otherwise" is mandatory because it rewards the agent in the case of no reward feature being active. This specific reward feature represents any situations not directly represented in the reward design space. The reward function structure is thus given by

$$R = \sum_{j \in J} s(j)\, w(j)\, \phi(j)\, P(j) \tag{5.12}$$

where $\phi(j)$ is a potential function representing the travel time on the edge associated with situation $j \in J$. This potential function is used because, in the TAP, the quality of a route is measured by its travel cost instead of by the number of edges (hops) as in CMOTP.

This experiment only uses the scenario ND (Section 5.2.2) in the evaluation because, here, it is wanted to find reward functions that best solve the learning task of an specific scenario—the use of reward functions that best solve the three TAP scenarios presented in Section 5.2.2 is discussed in Section 5.2.5.2. For comparison purposes, consider

Table 5.13: Reward features for Edge-based-QL.

J	Description
j_0	if the edge takes the agent directly to its destination vertex
j_1	if the edge brings the agent back to its origin vertex
j_2	if the edge was already traveled by the agent
j_3	otherwise

the difference rewards function R_{DR}, defined as in Equation 5.11.

Solving the EORP in scenario ND resulted in 26 Pareto-optimal ORFs from which it was selected a reward function R^* (as seen in Table 5.14) for comparison to the R_{DR}. In Table 5.14, it is possible to observe that R^* uses three features that reward the agent negatively. Only $\approx 4\%$ of the solutions use feature j_2. Regarding fitness, both R^* and R_{DR} reached solutions closer to the one obtained by the MSA (50.28). Thus, the collective of agents learning with each one of the functions in Table 5.14 is converging to policies close to the user equilibrium. However, a t-test conducted with 95% of confidence interval has shown that R^* overcomes R_{DR} regarding f_1. Regarding learning effort (f_2), R^* yield results that are slightly superior ($\approx 0.25\%$) to R_{DR}. This experiment showed that, even in this problem with 2000 agents, EORP is capable of providing solutions at least as good as those produced by the difference rewards function but without requiring a designer to manually determine that such a function (or a variant of it) is indeed the most appropriate. In other words, the space of solutions that may be found by solving EORP include well-known manually-constructed in the literature, but is more general in that it allows for more fine-tuned functions to be discovered if one exists that results in a better trade-off between the objectives specified by the designer. Also, the resulting traffic assignment is very close to the equilibrium solution identified by the MSA. Moreover, most importantly, the resulting ORFs do not make the assumptions about the availability of global information such as the difference rewards method.

The following experiment extends the EORP to the same assumptions made by the difference rewards method and checks if EORP can find functions that perform better than DR. For this, consider $\phi = G(z) - G(z_{-i})$ in the composition of the reward function. Now the traditional difference rewards function belongs to the reward design space of EORP— in particular, consider a reward function that is just composed by the feature j_3. By solving this optimization problem, EORP found 76 solutions from which the R^*_{DR} was selected for comparison purposes. Note in Table 5.14 that this function significantly reduces both fitness and learning effort when compared to R_{DR}.

All R^*, R_{DR} and R^*_{DR} have yield results that are close to the user equilibrium of

the TAP and very similar regarding fitness and learning effort. However, it is important to note that R^* does not assume the use of global information as in R_{DR} and R^*_{DR}. Besides this, this experiment also shows that by providing global information to the EORP, it can *automatically* identify a more efficient variant of the standard difference rewards function.

Table 5.14: Fitness (f_1) and effort (f_2) in TAP.

	$\mathbf{R}^*$		$\mathbf{R_{DR}}$	$\mathbf{R^*_{DR}}$	
$\mathbf{f_1}$	50.32 ± 0.04		50.35 ± 0.05	50.3 ± 0.07	
$\mathbf{f_2}$	4901.22 ± 2.88		4913.43 ± 2.89	4868.96 ± 2.25	
j_0	-0.23	88.43%	-	-0.25	100%
j_1	-0.57	23.8%	-	-	2.63%
j_2	-	3.85%	-	-0.44	97.37%
j_3	-0.34	100%	-	0.03	100%

5.2.5.2 Different Reward Feature Representations

Table 5.15 presents the set of reward features (column "J") manually extracted for edge and route-based-QL. Edge-based-QL has more reward features available than route-based-QL because edge-based-QL abstracts less environment information in its action structure. It provides an environment richer in agent and environment information, which can be used to generate reward features. The reward design space represented by these reward features captures the most common situations faced by the road users.

Table 5.15: Set of reward features (J) available for edge-based and route-based-QL. The column ϕ describes the function of each reward feature. ϕ is composed by variables present in scenario's cost functions (see Equations 5.2 - 5.5) but here its values are relative the edge or route that composes a given action a. For this reason, the identifier a is used in variables such as c_a, t^0_a, etc.

J	**Description**	ϕ	**Edge-based**	**Route-based**
j_0	edge takes the agent to its origin vertex?	c_a	✓	✗
j_1	edge takes the agent to its destination vertex?	c_a	✓	✗
j_2	edge was already traveled by the agent?	c_a	✓	✗
j_3	traveled distance	t^0_a	✓	✓
j_4	travel cost under free-flow condition	t^0_a	✓	✓
j_5	flow of vehicles	V_a	✓	✓
j_6	travel time	c_a	✓	✓

This experiment presents reward features with different representation. The reward features considered so far represent "true" or "false" expressions. In other words, such kind of reward feature can be activated or not in a given state. The reward features from j_0 to j_2 in Table 5.15 are good examples of such type of reward features. For instance, for an edge-based-QL agent, the reward feature j_0 is only "activated" when the

action taken by the agent leads it back to its initial state. In all other actions taken by such agent, this reward feature will be "deactivated" ($s\left(j_0\right) = 0$) in the environment.

The second type of reward features considered in this experiment represents reward features that are always present in any state. The reward features from j_3 to j_6, in Table 5.15, are good examples of such kind of reward features. Note that they are not represented by true or false statements. They, instead, are always present in agents' lifetime, independently of their actions. For instance, the reward feature j_3 represents the travel distance spent by an agent following its action. This reward feature is always activated ($s\left(j_3\right) = 1$) in the environment. If this reward feature is used to compose a given reward function, the reward signal produced by such a reward feature will be given according to the distance traveled by the agent in its last action. The traveled distance is given by the function ϕ.

The use of these two kind of reward features requires minor modifications to the EORP. Independently of a reward feature j having binary or real representation, its representation is abstracted by the functions $s\left(j\right)$ and $\phi\left(j\right)$. Table 5.15 uses a specific function ϕ, according to each reward feature. For instance, the function of j_3 is given by the travel distance function, while the function of j_5 is given by the flow of vehicles function. For the reward features with binary representation, the travel cost c_a of the executed action a (that can represent an edge or route depending on the learning algorithm being used) is used as a function because this function represents the goal of the learning task.

The reward design space of edge and route-based-QL is given by $\mathcal{R}\left(J\right) \subset \mathcal{R}^{|J|}$, where J is the set of reward features available for each algorithm. The reward function

$$R = \sum_{j \in J} s\left(j\right) w\left(j\right) P\left(j\right) \phi\left(j\right) \tag{5.13}$$

differs from the one used in Equation 5.12 because, in this new reward function structure, the reward signal is composed by all the reward features activated in a given state, rather than just by the first one activated in J, as in Equation 5.12.

Another characteristic that differentiates this experiment from the previous is the nature of designed reward functions. In the previous experiment, the reward functions are designed for best solving a specific scenario (scenario ND). The current experiment, by contrast, intends to find an optimal reward function that best solves the *three* TAP instances presented in Section 5.2.2. Therefore, each potential ORF is evaluated regarding fitness and effort produced in all three scenarios.

The multi-objective evaluation function used by both MARL algorithm is given by

$$\mathcal{F}(H) = [f_1(H), -f_2(H)] \tag{5.14}$$

where:

- $f_1(h_i) = -c(p_i)$ is the fitness function of each agent i, which represents the negative cost of its learned route p_i. This function considers only the fitness produced in a given episode. By using this function, the fitness is maximized when the travel time of the agent is minimized; and
- $f_2(h_i) = \frac{\sum_{\lambda=0}^{|\Lambda|} t_\lambda}{|\Lambda|}$ is the learning effort function, which represents the average number of decisions taken during the lifetime of the agent.

By solving the EORP for edge-based-QL, it produced 25 Pareto-optimal reward functions, from which the reward function in Table 5.16 was arbitrarily selected for comparison purposes. For route-based-QL, the EORP returned just a single reward function because in edge-based-QL the learning agents always performs just one action per episode, resulting in a learning effort always equals to 1.

Table 5.16: Optimal reward functions for edge-based and route-based-QL in the three TAP scenarios.

	Edge-based-QL			**Route-based-QL**		
Scenario	OW	ND	SF	OW	ND	SF
f_1	67,305	50,322	21,918	67,172	50,255	20,559
j_0		0.52			n/a	
j_1		-0.55			n/a	
j_2		-0.58			n/a	
j_3		-			-	
j_4		0			-0.28	
j_5		-			-	
j_6		-1			-0.61	

Table 5.17 presents the performance of edge-based-QL and route-based-QL under the guidance of reward functions obtained by ED, DR, and the EORP previously presented in Table 5.16. The reward functions are evaluated in the three scenarios discussed in Section 5.2.2. Two analysis are performed in this experiment. The first analyses compares the obtained results of all methods regarding obtained ATT. On the other hand, in the second analysis, the convergence curves of each algorithm are compared. The results concerning ATT for each approach as well as the ATT under UE condition are presented in Table 5.17.

Table 5.17: Average travel time and standard deviation (in parentheses).

Method	**Scenario**		
	OW	**ND**	**SF**
Edge-based-QL-EORP	67,305 (0,036)	50,322 (0,047)	21,918 (0,089)
Edge-based-QL-DR	*66,980 (0,025)*	50,349 (0,053)	21,880 (0,120)
Edge-based-QL-ED	67,150 (0,017)	50,236 (0,40)	21,940 (0,097)
Route-based-QL-EORP	67,172 (0,071)	50,255 (0,005)	*20,559 (0,059)*
Route-based-QL-DR	*66,991 (0,118)*	*50,028 (0,002)*	20,729 (0,098)
Route-based-QL-ED	67,156 (0,014)	50,256 (0,006)	*20,594 (0,067)*
User equilibrium	≈ 67.163	≈ 50.277	≈ 20.785

In the OW scenario, all methods achieved ATTs closer to the one obtained under UE condition. However, for both learning algorithms, the DR reward function yielded results slightly better. Since the OW scenario has few options of routes to the two destination nodes in its OD-matrix (nodes L and M), it creates a lot of competition among selfish-agents for specific parts of the road network. In such situations, the DR function is capable of achieving better ATTs because it elicits cooperation among agents. Such cooperation reduces the competition for the most attractive edges and benefits the system as a whole.

For the ND scenario, again all approaches achieved ATTs closer to the one under UE solution. The DR function achieved ATT better than UE just in route-based-QL because this scenario is more challenging than the OW one: the network topology and demand of scenario ND are larger than in the OW scenario, which makes the learning task harder to edge-based-QL due to the larger search space and more noisy reward signals.

In the SF scenario, the learning algorithm played a major role in the ATT than the reward function. Regardless of the reward function used, route-based-QL was able to yield ATT at least as good as the one obtained under UE. For EORP and DR reward functions, the ATTs were even better than the one under UE. The massive demand and number of OD-pairs make the reward signals extremely noisy for the agents. However, even under such circumstances, the search space restriction provided by route-based-QL made it possible for the agents to learn more appropriate routes. For edge-based-QL, the three reward functions yielded ATTs worse than the ones obtained by route-based-QL. This is because the larger network topology of scenario SF increases the search space in the case of edge-based-QL.

The results presented so far indicate that with respect to the achieved ATT, the three reward functions are robust in solving the TAP. In larger scenarios, as in the SF presented here, independent of the reward function to be used, the edge-based-QL ends

up limiting the learning capacity of agents. It is harder for the learning agents to learn optimal policies in the larger and highly dynamic search space provided by edge-based-QL. To overcome such limitation, the results in this thesis suggest that in larger scenarios it is more appropriate to use route-based-QL than edge-based-QL.

In order to better evaluate the reward functions discussed so far, the learning curves—ATT convergence along episodes—of the learning methods and reward functions are presented in Figure 5.7 (for the OW scenario), Figure 5.8 (for the ND scenario), and Figure 5.9 (for the SF scenario). In the three scenarios, it is possible to observe that route-based-QL converges faster than edge-based-QL independently of the reward function being used. In the early episodes, edge-based-QL agents have no knowledge of their MDP, and the exploration rate is very high. Consequently, agents have a high probability of converging to undesired behaviors (routes). Despite this, after some episodes, the learning curves of both methods converge to similar solutions in all scenarios.

In the first experiment, it has been shown that the three used reward functions are robust in guiding the learning process of TAP agents. In small scenarios such as OW and ND, the DR function was able to yield slightly better ATTs. Despite this, it was evident that the learning algorithm has considerable more influence on the performance of the agents than the reward functions used. The second experiment also concludes that all reward functions have similar performance regarding convergence.

The three reward functions evaluated here are very similar concerning their obtained ATT and convergence time. However, EORP and DR reward functions are not so trivial to be applied in TAP. EORP reward functions require much work during the definition and resolution of the EORP, while DR functions assume a communication channel among agents by which they obtain the necessary information for the calculation of the system's performance. The expert-designed reward function seems to be a more adequate reward function for the TAP because its definition is more straightforward than others.

Figure 5.7: Performance vs. time on scenario OW.

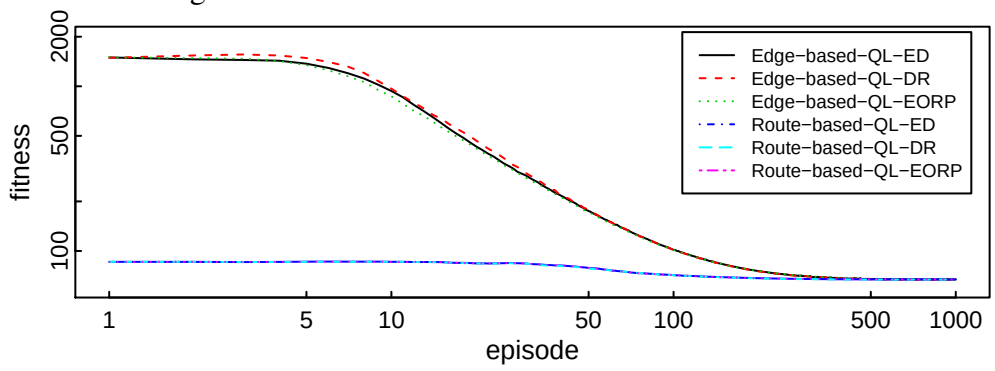


Figure 5.8: Performance vs. time on scenario ND.

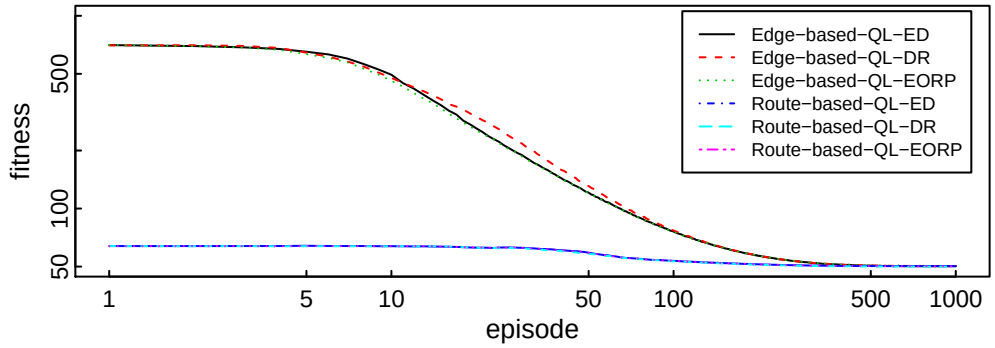
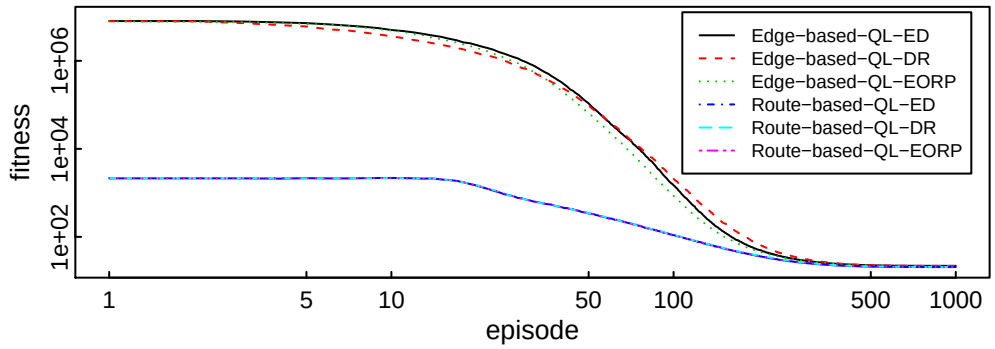


Figure 5.9: Performance vs. time on scenario SF.



5.3 Discussion

The EORP was evaluated in two different domains. The first domain, called the coordinated multi-agent object transportation problem, provides a task with two agents (a single-agent variant of this problem was also considered). Regarding this domain, this thesis showed that:

- The use of multiple goal functions (fitness, learning effort and so on) in the evaluation of optimal reward functions provided by EORP is beneficial for the agents because it results in agents that learn optimal policies faster. A reward design space may present several reward functions that yield an optimal policy for a given problem. The traditional ORP is not able to find the one that results in faster learning because ORP does not take into account the trade-off between multiple designer goals. In EORP, the trade-off of multiple designer goal functions is considered. For this reason, EORP shows to be superior to ORP in the finding optimal reward functions.

- The automatic selection of reward features is helpful for the system's designer. Selecting the most appropriate set of reward features to compose a reward function is not an easy task due to a large number of potential reward features the problem may provide. By giving an appropriate set of reward features to the EORP, it was able to

find optimal reward functions that overcome the expert-designed reward functions of CMTOP regarding both fitness and learning effort.

- Optimizing a single reward function that is used by a set of agents with a common task scales better than by optimizing one reward function per learning agent. Optimizing a single reward function per set of agents results in an optimization problem with lower dimensionality, which is easier to be solved.

- EORP is general enough to deal with both single and multi-agent reinforcement learning problems.

The second domain, called the traffic assignment problem, provides non-cooperative tasks that involve thousands of learning agents. In this domain, the present thesis showed that:

- The performance of EORP designed reward functions is at least as good as the one obtained by the difference rewards function. However, EORP reward functions do not make use of global information in its internal structure (i.e., reward features and reward signals only use information observable to the agent), as occurs in the difference rewards function. The EORP was able to find in its rich reward design space reward functions that result in the desired behavior, but without making use of global information.

- By providing the same assumptions of use of global information in the internal structure of the reward function to the EORP, EORP was shown to be able to find reward functions that overcome the traditional difference rewards function in terms of both fitness and learning effort. This finding also suggests that the traditional difference rewards function may not be the best reward function to stimulate cooperative behavior between agents.

- EORP deals with the different representation of reward features. A reward design space that contains reward features that have both binary and numeric representation was provided to the EORP. The method was able to combine both types of reward features in optimal reward functions that, when optimized by agents, resulted in solutions closer to the optimum for the evaluated scenarios. It enables that the EORP to deal with the diversity of state information present in application domains.

6 FINAL REMARKS

6.1 Conclusions

This thesis presented a novel and general method for reward function design in the context of reinforcement learning. A reward function is the element in the reinforcement learning framework that defines the goal of learning agents. The designer of the reinforcement learning system is the responsible for defining a reward function that, when optimized by learning agent(s), results in agents learning a desired behavior. The literature presents no general rule for the designing reward functions as well as no guarantees that the designed reward function is the most appropriated for a given task. The use of an inappropriate reward function may lead the agent to learning inappropriate behaviors and, consequently, rendering ineffective the application of reinforcement learning in such a problem.

The literature presents some methods to assist the designer in the task of designing effective reward functions. Existing work on the *Optimal Reward Problem* (ORP) propose mechanisms to automatically design reward functions. However, their application is limited to specific sub-classes of single or multi-agent reinforcement learning problems. Moreover, such methods identify which rewards should be given in which situation, but not which aspects of the state or environment should be used when defining a reward function.

The method proposed in this thesis is called *extended*-optimal reward problem (EORP) because it is strongly inspired in the ORP. The EORP is a more complete and also more versatile version of the ORP, in which the main limitations of the traditional ORP are addressed. The EORP provides four main contributions compared to the traditional ORP. All contributions were experimentally validated on two different domains: the object transportation problem and the traffic assignment problem.

The first contribution of this thesis is the automatic selection and identification of situations (or state features) to reward. Existing approaches consider that the designer defines all the situations in which the agents must be rewarded, and the method only adjusts the reward signal for each situation. However, some problems may present many situations that could *potentially* contribute to the reward function. If the designer chooses an inappropriate set of reward features, that can impact the quality of the reward function found by solving the ORP. The EORP can automatically identify both situations and re-

ward signals that are relevant for the optimal reward function. The designer should just provide the set of potential situations to reward as input. Through experiments in the object transportation domain, this thesis shows that agents can benefit from the correct selection of reward features provided by the EORP. Reward functions obtained by the EORP were capable of improving both the quality of the learned behavior as well as the learning effort when compared to an expert-designed reward function.

The second contribution of this thesis is the generality of application of our formulation for dealing with RL problems that may be either single and multi-agent setting. In the object transportation domain, it was demonstrated that the EORP was capable of designing reward functions that lead the agents to learning optimal policies in both single and multi-agent cooperative settings. The applicability of EORP in multi-agent competitive tasks with more than 2 agents was demonstrated in the traffic assignment domain.

The third contribution of this thesis is associated with scalability issues. Existing works on ORP optimize a private reward function for each learning agent in order to deal with a specific task. It was shown that in problems with a large number of agents, the optimization might not converge to desired solutions due to the slow convergence of optimization methods in high-dimensional search spaces. EORP scales well because it optimizes a unique reward function for all agents that share a common task. Experiments performed in the traffic assignment domain showed that EORP was able to find effective reward functions even in the presence of thousands of agents because the dimensionality of the search problem in EORP is not dependent of the number of learning agents.

The fourth contribution of this thesis is related to the evaluation of a reward function. The space of reward functions may contains multiple reward functions that produce the same behavior, but that differ in the learning effort required to acquire such an optimal policy. The EORP takes into account the trade-off between fitness and learning effort spent in the learning process. Therefore, functions found by the EORP aim at producing the best behavior (fitness) in the best learning time (effort). Experiments conducted in the object transportation domain showed that by considering multiple designer goal-functions, the EORP is capable of providing agents that converge faster to optimal policies. In the object transportation domain, this thesis compared the reward functions provided by EORP against an expert-defined reward function and showed that EORP was capable of providing agents with better learning capabilities. In the traffic assignment domain, such a comparison was extended to compare EORP against a method present in the literature called difference rewards. Experiments performed in different traffic assignment

scenarios show that the EORP can find reward functions that overcome the difference rewards technique in performance but without making use of global information. When the same assumptions made by difference rewards are provided to EORP, EORP was also capable of significantly improving both fitness and learning effort with respect to the classic DR method.

6.2 Future Work

As future work, the present thesis outlines the following directions:

- Propose a theoretical analysis of the EORP. This thesis extensively evaluates the EORP, but just experimentally. No theoretical analysis of the model has been performed so far. In future work, two theoretical analysis are intended. The first aims at finding (if possible) the convergence bounds of EORP. The second analysis concerns the complexity analysis of time and memory of EORP.

- Investigate the choice of optimization solver. This thesis used the CMA-ES algorithm for solving EORP because it has been shown successfully in solving many applications in the literature (and also in the experiments of this thesis). However, several other solvers such as genetic algorithms (FastPGA, NSGAII, NSGAIII) or multiobjective particle swarm optimization could be used as EORP solver. It is part of the future work of this thesis to provide analyses about the right selection of the optimization solver as well as a runtime analysis.

- Investigate the effects of turning EORP into a single-objective optimization problem. The EORP is modeled as a multi-objective optimization problem that considers the trade-off among multiple goal-functions (fitness, learning effort, etc.) defined by the system's designer. This strategy adopted in EORP gives more flexibility to the system's designer during the selection of which optimized reward function to be used in a given problem. However usually the system's designer uses some ranking to select among a set of optimized reward functions. By converting the EORP into a single-objective optimization problem that considers some raking among the multiple-objectives, the EORP can explore the use of more efficient solvers. The advantages and disadvantages of using this strategy will be better investigated ad a future work.

- Automatic extraction of reward features. The EORP considers that the system's designer provides the EORP with a set with reward features that can potentially compose a reward function. It is also assumed that the system's designer has enough domain knowledge to extract such a set of reward features. The EORP selects from such set the one(s) that will compose the reward function. A future work of this thesis intends to investigate the development of an EORP that, given the observations of an agent and environment variables, could automate the extraction of potential reward features.

REFERENCES

ABBEEL, P. et al. Apprenticeship learning for motion planning with application to parking lot navigation. In: **2008 IEEE/RSJ International Conference on Intelligent Robots and Systems**. [S.l.]: IEEE, 2008. p. 1083–1090. ISBN 978-1-4244-2057-5.

ABBEEL, P.; NG, A. Y. Apprenticeship learning via inverse reinforcement learning. In: **Twenty-first international conference on Machine learning - ICML '04**. New York, New York, USA: ACM Press, 2004. p. 1. ISBN 1581138285.

AGOGINO, A.; TUMER, K. Multi–agent reward analysis for learning in noisy domains. In: DIGNUM, F. et al. (Ed.). **AAMAS '05: Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems**. New York, NY: [s.n.], 2005. II, p. 81–88.

BARTO, A. G.; SINGH, S.; CHENTANEZ, N. Intrinsically motivated learning of hierarchical collections of skills. In: **Proc. 3rd Int. Conf. Development Learn**. [S.l.: s.n.], 2004. p. 112–119.

BAZZAN, A. L. C.; GRUNITZKI, R. A multiagent reinforcement learning approach to en-route trip building. In: **2016 International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2016. p. 5288–5295.

BUREAU, o. P. R. **Bureau of Public Roads: Traffic Assignment Manual**. [S.l.], 1964.

BUŞONIU, L.; BABUSKA, R.; SCHUTTER, B. D. A comprehensive survey of multiagent reinforcement learning. **Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on**, IEEE, v. 38, n. 2, p. 156–172, 2008.

BUŞONIU, L.; BABUŠKA, R.; SCHUTTER, B. D. Multi-agent reinforcement learning: An overview. In: **Innovations in Multi-Agent Systems and Applications-1**. [S.l.]: Springer, 2010. p. 183–221.

CLAUS, C.; BOUTILIER, C. The dynamics of reinforcement learning in cooperative multiagent systems. In: **Proceedings of the Fifteenth National Conference on Artificial Intelligence**. [S.l.: s.n.], 1998. p. 746–752.

DEVLIN, S.; KUDENKO, D. Dynamic potential-based reward shaping. In: **Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems**. [S.l.]: International Foundation for Autonomous Agents and Multiagent Systems, 2012. p. 433–440.

DEVLIN, S. et al. Potential-based difference rewards for multiagent reinforcement learning. In: INTERNATIONAL FOUNDATION FOR AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS. **Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems**. [S.l.], 2014. p. 165–172.

GRUNITZKI, R.; BAZZAN, A. L. C. Combining car-to-infrastructure communication and multi-agent reinforcement learning in route choice. In: BAZZAN, A. L. C. et al. (Ed.). **Proceedings of the Ninth Workshop on Agents in Traffic and Transportation (ATT-2016)**. New York: CEUR-WS.org, 2016. ISSN 1613-0073. Disponível em: <http://ceur-ws.org/Vol-1678/paper12.pdf>.

GRUNITZKI, R.; BAZZAN, A. L. C. Comparing two multiagent reinforcement learning approaches for the traffic assignment problem. In: **Intelligent Systems (BRACIS), 2017 Brazilian Conference on**. [S.l.: s.n.], 2017.

GRUNITZKI, R.; RAMOS, G. d. O.; BAZZAN, A. L. C. Individual versus difference rewards on reinforcement learning for route choice. In: **Intelligent Systems (BRACIS), 2014 Brazilian Conference on**. [S.l.: s.n.], 2014. p. 253–258.

GRUNITZKI, R.; RAMOS, G. d. O.; BAZZAN, A. L. C. Uma ferramenta para alocação de tráfego e aprendizagem de rotas em redes viárias. In: **Anais do XXVIII Congresso de Pequisa e Ensino em Transportes (ANPET 2014)**. [s.n.], 2014. ISBN 978-85-87893-17-8. Disponível em: <www.inf.ufrgs.br/maslab/pergamus/pubs/grunitzki+2014-anpet.pdf>.

GRUNITZKI, R.; SILVA, B. C. da; BAZZAN, A. L. C. A flexible approach for designing optimal reward functions. In: DAS, S. et al. (Ed.). **Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)**. São Paulo: IFAAMAS, 2017. p. 1559–1560. Disponível em: <http://ifaamas.org/Proceedings/aamas2017/pdfs/p1559.pdf>.

GRUNITZKI, R.; SILVA, B. C. da; BAZZAN, A. L. C. Towards designing optimal reward functions in multi-agent reinforcement learning problems. In: **Proc. of the 2018 International Joint Conference on Neural Networks (IJCNN 2018)**. Rio de Janeiro: [s.n.], 2018.

HU, J.; WELLMAN, M. P. Nash q-learning for general-sum stochastic games. **J. Mach. Learn. Res.**, v. 4, p. 1039–1069, 2003.

IGEL, C.; HANSEN, N.; ROTH, S. Covariance matrix adaptation for multi-objective optimization. **Evolutionary computation**, MIT Press, v. 15, n. 1, p. 1–28, 2007.

JENNINGS, N. R. On agent-based software engineering. **Artificial Intelligence**, v. 117, n. 2, p. 277 – 296, 2000. ISSN 0004-3702. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0004370299001071>.

KOLTER, J. Z.; ABBEEL, P.; NG, A. Y. Hierarchical apprenticeship learning with application to quadruped locomotion. In: PLATT, J. C. et al. (Ed.). **Advances in Neural Information Processing Systems 20**. [S.l.]: Curran Associates, Inc., 2008. p. 769–776.

LIU, B. et al. Optimal Rewards for Cooperative Agents. **Autonomous Mental Development, IEEE Transactions on**, v. 11, n. 4, 2014. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6920028>.

LIU, B. et al. Optimal rewards in multiagent teams. In: **2012 IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL 2012**. [S.l.: s.n.], 2012. ISBN 9781467349635.

MOMBAUR, K.; TRUONG, A.; LAUMOND, J.-P. From human to humanoid locomotion? An inverse optimal control approach. **Autonomous Robots**, v. 28, n. 3, p. 369–383, apr 2009. ISSN 0929-5593.

NG, A. Y.; HARADA, D.; RUSSELL, S. Policy invariance under reward transformations: Theory and application to reward shaping. In: **In Proceedings of the Sixteenth International Conference on Machine Learning**. [S.l.]: Morgan Kaufmann, 1999. p. 278–287.

NGUYEN, S.; DUPUIS, C. An efficient method for computing traffic equilibria in networks with asymmetric transportation costs. **Transportation Science**, Informs, v. 18, n. 2, p. 185–202, 1984.

NIEKUM, S.; BARTO, A.; SPECTOR, L. Genetic Programming for Reward Function Search. **IEEE Transactions on Autonomous Mental Development**, v. 2, n. 2, p. 83–90, 2010. ISSN 1943-0604.

ORTÚZAR, J. d. D.; WILLUMSEN, L. G. **Modelling transport**. 4. ed. Chichester, UK: John Wiley & Sons, 2011.

RAMOS, G. de. O.; GRUNITZKI, R. An improved learning automata approach for the route choice problem. In: KOCH, F.; MENEGUZZI, F.; LAKKARAJU, K. (Ed.). **Agent Technology for Intelligent Mobile Services and Smart Societies**. [S.l.]: Springer Berlin Heidelberg, 2015, (Communications in Computer and Information Science, v. 498). p. 56–67. ISBN 978-3-662-46240-9.

RAMOS, G. de. O.; GRUNITZKI, R.; BAZZAN, A. L. C. On improving route choice through learning automata. In: **Proceedings of the Fifth International Workshop on Collaborative Agents – Research & Development (CARE 2014)**. [s.n.], 2014. p. 1–12. Disponível em: <http://www.inf.ufrgs.br/maslab/pergamus/pubs/Ramos+2014care.pdf>.

RATLIFF, N. D.; BAGNELL, J. A.; ZINKEVICH, M. A. Maximum margin planning. In: **Proceedings of the 23rd international conference on Machine learning - ICML '06**. New York, New York, USA: ACM Press, 2006. p. 729–736. ISBN 1595933832. ISSN 17458358.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. Third. [S.l.]: Prentice Hall, 2010. (Prentice Hall series in artificial intelligence). ISBN 9780136042594.

SINGH, S.; LEWIS, R. L.; BARTO, A. G. Where do rewards come from. In: **Proceedings of the annual conference of the cognitive science society**. [S.l.: s.n.], 2009. p. 2601–2606.

SINGH, S. et al. Intrinsically motivated reinforcement learning: An evolutionary perspective. **Autonomous Mental Development, IEEE Transactions on**, IEEE, v. 2, n. 2, p. 70–82, 2010.

SINGH, S. P.; BARTO, A. G.; CHENTANEZ, N. Intrinsically motivated reinforcement learning. In: **Advances in Neural Information Processing Systems 17 (NIPS 2004)**. [S.l.: s.n.], 2004. v. 17, p. 1281–1288.

SORG, J.; LEWIS, R.; SINGH, S. Internal Rewards Mitigate Agent Boundedness. In: **Proceedings of the 27th International Conference on Machine Learning (ICML-10)**. [S.l.: s.n.], 2010. p. 1007–1014. ISBN 9781605589077.

SORG, J.; LEWIS, R. L.; SINGH, S. P. Reward Design via Online Gradient Ascent. In: LAFFERTY, J. D. et al. (Ed.). **Advances in Neural Information Processing Systems 23**. [S.l.]: Curran Associates, Inc., 2010. p. 2190–2198.

SORG, J. D. **The optimal reward problem: Designing effective reward for bounded agents**. Tese (Doutorado) — The University of Michigan, 2011.

SUTTON, R.; BARTO, A. **Reinforcement Learning: An Introduction**. Cambridge, MA: MIT Press, 1998.

SYED, U.; SCHAPIRE, R. E. A Game-Theoretic Approach to Apprenticeship Learning. **Advances in Neural Information Processing Systems 20**, v. 20, p. 1–8, 2008.

TUMER, K.; AGOGINO, A. Distributed agent-based air traffic flow management. In: **Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems**. New York, NY, USA: ACM, 2007. p. 1–8. ISBN 978-81-904262-7-5.

TUMER, K.; WOLPERT, D. A survey of collectives. In: TUMER, K.; WOLPERT, D. (Ed.). **Collectives and the Design of Complex Systems**. [S.l.]: Springer, 2004. p. 1–42.

WARDROP, J. G. Some theoretical aspects of road traffic research. **Proceedings of the Institution of Civil Engineers, Part II**, v. 1, n. 36, p. 325–362, 1952.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, Kluwer Academic Publishers, Hingham, MA, USA, v. 8, n. 3, p. 279–292, 1992. ISSN 0885-6125.

WIEWIORA, E.; COTTRELL, G.; ELKAN, C. Principled methods for advising reinforcement learning agents. In: **Proceedings of the Twentieth International Conference on Machine Learning**. [S.l.: s.n.], 2003. p. 792–799.

WOOLDRIDGE, M. J. **An Introduction to MultiAgent Systems**. Chichester: John Wiley & Sons, 2009. 461 p. Second edition.

YEN, J. Y. Finding the k shortest loopless paths in a network. **Management Science**, v. 17, n. 11, p. 712–716, 1971. Disponível em: <http://pubsonline.informs.org/doi/abs/10.1287/mnsc.17.11.712>.

ZHIFEI, S.; Er Meng Joo. A review of inverse reinforcement learning theory and recent advances. In: **2012 IEEE Congress on Evolutionary Computation**. [S.l.]: IEEE, 2012. p. 1–8. ISBN 978-1-4673-1509-8.

ZIEBART, B. D. et al. Maximum entropy inverse reinforcement learning. **AAAI Conference on Artificial Intelligence**, p. 1433–1438, 2008. ISSN 10450823.

ZINKEVICH, M. Online convex programming and generalized infinitesimal gradient ascent. In: **In Proceedings of the Twentieth International Conference on Machine Learning**. Menlo Park, USA: AAAI Press, 2003. p. 928–936.

APPENDIX A — UMA ABORDAGEM FLEXÍVEL PARA RECOMPENSAS ÓTIMAS EM PROBLEMAS DE APRENDIZADO POR REFORÇO MULTIAGENTE

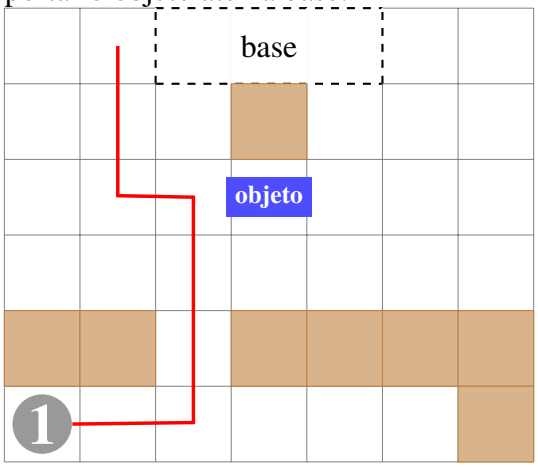
O aprendizado por reforço lida com problemas onde um agente tenta aprender um comportamento através de sucessivas iterações com o ambiente (SUTTON; BARTO, 1998). O comportamento do agente é representado por uma política que mapeia estados do ambiente para ações, com base na expectativa de recompensa a ser recebida em cada par estado-ação. A qualidade de uma ação tomada por um agente durante o processo de aprendizagem por reforço é avaliada com base em um sinal numérico, conhecido como *recompensa*, recebido do ambiente após o agente ter executado tal ação. A função de recompensa determina o valor do sinal de recompensa que um agente irá receber do ambiente. É responsabilidade do projetista[1] do sistema baseado em aprendizado por reforço especificar uma função de recompensa que, quando otimizada por um agente, resulte no aprendizado de uma política ótima—a qual representa a forma com que o agente deve se comportar no ambiente. O desempenho do agente ou coletivo de agentes (quando o problema possui mais de um agente aprendendo simultaneamente, chamamos de aprendizado por reforço multiagente - MARL) aprendendo através de aprendizado por reforço, está diretamente relacionado com a função de recompensa sendo otimizada por ele(s), pois este é o mecanismo responsável por definir o objetivo da tarefa de aprendizagem.

O uso de funções de recompensa efetivas proporciona benefícios aos agentes que vão além da aquisição de uma política ótima. Múltiplas funções de recompensa, quando otimizadas, podem produzir uma mesma política ótima, porém com diferente esforço de aprendizagem ao agente. Considere como exemplo a tarefa episódica apresentada na Figura A.1. Nesta tarefa, introduzida inicialmente por (BUŞONIU; BABUŠKA; SCHUTTER, 2010), o agente 1 deve encontrar o objeto localizado no labirinto, agarra-lo e transporta-lo até a base (uma completa descrição desta tarefa é apresentada na Seção 5.1). A solução ótima para a tarefa é representada pela linha vermelha, a qual resolve a tarefa em apenas 8 passos/movimentos. Os autores do cenário apresentam a função de recompensa definida na Equação A.1 para resolver a tarefa. Esta função consegue guiar o agente no processo de aprendizagem da solução da tarefa em tempo ótimo. Apesar da existência desta função proposta pelos autores do cenário, muitas outras funções de recompensa,

[1] Nesta tese, sempre que nos referimos ao "projetista", nós não estamos necessariamente nos referindo a uma única pessoa, mas ao time que está desenvolvendo a solução baseada em aprendizado por reforço.

como a apresentada na Equação A.2, são capazes de guiar o aprendizado de uma política ótima para esta tarefa.

Figure A.1: Domínio de transporte de objetos. A tarefa de aprendizagem consiste em um agente aprender a transportar o objeto até na base.



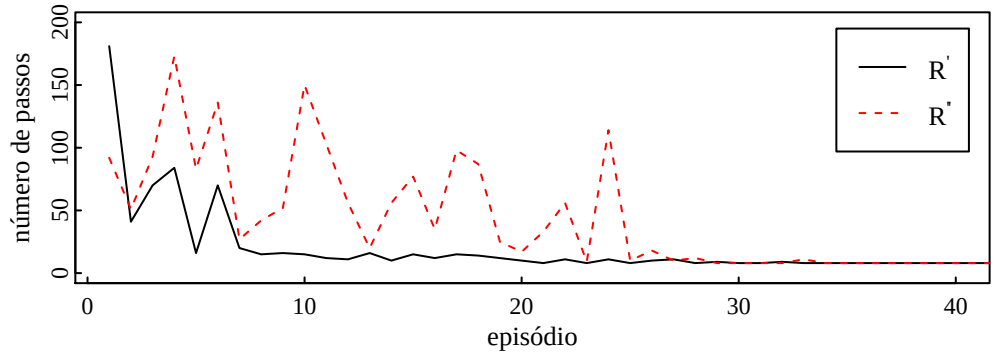
$$R' = \begin{cases} 1, & \text{se o objeto chegou na base} \\ 0.1, & \text{se o objeto foi agarrado} \\ 0, & \text{caso contrário} \end{cases} \tag{A.1}$$

$$R'' = \begin{cases} 1, & \text{se o objeto chegou na base} \\ 0, & \text{caso contrário} \end{cases} \tag{A.2}$$

Na Figura A.2, nós podemos observar o desempenho do agente 1 aprendendo com ambas as funções R' e R''. Nos episódios finais, ambas as funções se mostram capazes de resolver a tarefa no tempo mínimo. Porém, agora vamos considerar uma segunda métrica de desempenho, chamada *esforço de aprendizagem*. O esforço de aprendizagem para adquirir uma política ótima nesta tarefa é representada pelo número acumulativo de passos (ou decisões) gastos pelo agente ao longo dos episódios de sua vida. Se nós também considerarmos o esforço de aprendizagem, as funções de recompensa R' e R'' deixarão de ser equivalentes. Ao aprender com a função de recompensa R' o agente gasta ≈ 50 menos passos ao longo de sua vida quando comparado a função R''.

Para este cenário que utilizamos como exemplo, pode ser que seja mais interessante para o projetista utilizar a função R' ao invés da função R'', já que ela converge mais rapidamente. No entanto, vale argumentar se R' é realmente a função de recompensa mais adequada para esta tarefa? Não há uma resposta direta para esta pergunta porque existem muitas outras funções de recompensa capazes de representar a tarefa de aprendizagem

Figure A.2: Desempenho do agente 1 ao resolver o problema de transporte coordenado de objeto com as funções R' e R''.



em questão. Na Tabela A.1, nós apresentamos algumas das situações hipotéticas em que o projetista poderia considerar ser interessante (para o sucesso da tarefa) recompensar o agente. Em uma situação como essa, encontrar uma função de recompensa adequada não é uma tarefa simples, pois o espaço de funções de recompensa disponíveis é composto por qualquer combinação destas situações e suas respectivas recompensas.

Table A.1: Situações hipotéticas para recompensar o agente 1.

#	**Situação**
1	se o objeto chegou na base
2	se o agente agarrou o objeto
3	se o agente bateu na parede
4	se o agente tentou ocupar uma célula ocupada por um objeto
5	se o agente escolheu ficar na mesma célula

Como demonstrado no exemplo do transporte coordenado de objeto, definir a função de recompensa mais adequada não é uma tarefa fácil para o projetista. Embora seja assumido que o projetista tem conhecimento suficiente para projetar funções de recompensa para os agentes, o desempenho resultante é sensível às escolhas básicas tomadas pelo projetista em questões relacionadas ao projeto de funções de recompensa, como:

- em quais situações recompensar?
- quanto recompensar em cada situação?

Além destas questões, dependendo das características da tarefa de aprendizagem, outras questões ainda podem aparecer. Por exemplo, problemas de aprendizagem por reforço monoagente costumam apresentar menos questões associadas com o projeto de funções de recompensa do que problemas de caráter MARL. Em problemas MARL, a estratégia para a modelagem de função de recompensa utilizada pelo projetista pode mudar de acordo com o tipo de tarefa de aprendizagem—cooperativa, competitiva ou mista—e também

pela quantidade de agentes aprendendo. Por exemplo, em tarefas multiagente de caráter competitivo, o uso de ações conjuntas e recompensa de time pode ser mais interessante desde que o problema tenha poucos agentes, pois em cenários com grandes quantidades de agentes esta estratégia pode sofrer com o aumento da dimensionalidade (BUŞONIU; BABUSKA; SCHUTTER, 2008).

Uma função de recompensa inadequada pode levar o agente ao aprendizado de um comportamento inapropriado. Além disso, mesmo que uma função de recompensa possa guiar o agente a aprender uma política ótima, pode ser que o esforço de aprendizagem gasto—no que se refere a esforço computacional ou de tempo—para aprender tal política, torne o uso da abordagem baseada em aprendizado por reforço infactível. Durante o estágio de modelagem de uma problema de aprendizado por reforço, o projetista do sistema geralmente adota a seguinte estratégia:

- recompensar positivamente as ações que levam o agente a estados desejados;
- recompensar negativamente as ações que levam o agente a estados indesejados.

Tal estratégia estimula comportamentos dirigidos por motivações extrínsecas, i.e., motivações geradas por recompensas imediatas diretamente relacionadas aos objetivos do agente (BARTO; SINGH; CHENTANEZ, 2004). Porém, Singh, Lewis and Barto (2009), Singh et al. (2010) argumentam que esta estratégia pode não ser a melhor. Os autores mostram que os agentes podem ser beneficiados ao serem recompensados em estados intermediários, ou seja, estados que não estão diretamente relacionados ao objetivo da tarefa, porque isso instiga comportamentos impulsionados por curiosidade, novidade, surpresa e outras características internamente mediadas que são normalmente associadas com motivações intrínsecas.

Para projetar funções de recompensa com estas características, Singh, Lewis and Barto (2009) introduziram o *optimal reward problem* (em português, problema de recompensa ótima - ORP). Um ORP é composto e especificado por duas funções: i) uma função de recompensa R que guia o processo de aprendizagem do agente; e ii) uma função de fitness F que avalia a qualidade do comportamento aprendido pelo agente. A função de fitness representa os desejos do projetista do sistema na tarefa de aprendizagem em questão. Resolver o ORP consiste em encontrar a função de recompensa ótima (ORF), R^*, que maximiza a função de fitness. Em Singh et al. (2010), o ORP é aproximadamente resolvido através de uma estratégia de busca baseada em força bruta. Trabalhos subsequentes a este (SORG; LEWIS; SINGH, 2010b; NIEKUM; BARTO; SPECTOR, 2010;

LIU et al., 2012; LIU et al., 2014) propuseram métodos automáticos e mais eficientes para lidar com o ORP em problemas de caráter tanto mono quanto multiagente. Estes métodos apresentam diversas limitações, as quais restringem a sua aplicação à tarefas de aprendizagem bastante específicas.

Além destes dos métodos baseados em ORP, ainda existem algumas estratégias não baseadas em ORP que podem ser utilizadas (ao menos parcialmente) para lidar com funções de recompensa. A técnica chamada *difference rewards*, apresentada por Tumer and Agogino (2007), estimula a cooperação entre um coletivo de agentes através de modificações na estrutura básica da função de recompensa. Entretanto, o seu uso está limitado a tarefas cooperativas, onde é necessário ter completa observação do sistema. Outra estratégia bastante conhecida é o RL inverso (IRL), o qual identifica a função de recompensa que produz um comportamento que foi previamente observado por um especialista. Não é possível generalizar o uso de IRL em função da necessidade de um conjunto suficientemente grande de observações. Por fim, técnicas baseadas em *reward shaping* (NG; HARADA; RUSSELL, 1999) visam a aceleração do processo de aprendizagem através do fornecimento de conhecimento de domínio na forma de sinais adicionais de recompensa.

Cada um dos métodos anteriormente mencionados possui suas próprias limitações que impedem que eles sejam aplicados em outros tipos de problemas, além dos quais eles foram projetados. A literatura apresenta espaço para um método geral que automatize a tarefa de projeto de funções de recompensa nos mais diversos tipos de problema de aprendizagem. Para preencher esta lacuna, nesta tese, nós pretendemos responder a seguinte questão de pesquisa:

Questão de Pesquisa.

Existe um método que automatize o processo de projeto de funções de recompensa e que possa ser aplicado nas mais diversas tarefas de aprendizagem?

A nossa ideia de *método automático* se refere a um método que, dado um conjunto de entradas previamente definidas pelo projetista do sistema, o método possa encontrar automaticamente a(s) função(ões) de recompensa mais apropriada(s) para a tarefa de aprendizagem. Quando nos referimos as *mais diversas tarefas de aprendizagem*, nós estamos nos referindo a problemas mono e multiagente e suas variantes.

Entre os métodos existentes, o que melhor se adequa a nossa questão de pesquisa é o ORP, pois ele já foi desenvolvido para encontrar a melhor função de recompensa

que atenda as preferências do projetista do sistema. Com base no ORP, nesta tese, nós endereçamos a nossa questão de pesquisa através da seguinte hipótese:

Hipótese de Pesquisa.

Fornecendo as modificações necessários ao optimal reward problem, é possível proporcionar um método automático que pode lidar com projeto de funções de recompensa nos mais diversos tipos de problemas de aprendizagem por reforço

Para validar esta hipótese, nesta tese nós propomos uma abordagem flexível para projetar funções de recompensa efetivas, chamada ORP estendido (em inglês, extended-optimal reward problem - EORP). O termo *flexível* refere-se a ampla gama de problemas mono e multiagente no qual a nossa abordagem pode ser aplicada. Já o termo *efetivo* significa encontrar funções de recompensa que, quando otimizadas, resultam em rápida aquisição de política ótima.

A.1 Visão Geral do EORP

O EORP, inicialmente apresentado em (GRUNITZKI; SILVA; BAZZAN, 2017), endereça as principais limitações dos métodos existentes. A seguir, apresentamos como tais limitações são endereçadas no EORP.

A.1.1 Seleção automática de *features*

Uma *feature* em projeto de funções de recompensa representa uma situação em que o agente possa ser recompensado. As abordagens existentes consideram que o projetista do sistema conhece e define todas as situações em que os agentes devem ser recompensados. O método, por sua vez, apenas se encarrega de definir o sinal de recompensa recebido em cada uma destas situações. Uma desvantagem desta estratégia é que as tarefas de aprendizagem podem apresentar muitas situações onde o agente pode potencialmente ser recompensado. Se o projetista escolher um conjunto inadequado de features, isso pode impactar negativamente na qualidade do comportamento aprendido pelo agente.

O EORP diminui a responsabilidade do projetista em definir esse conjunto de features. Ele permite que o projetista defina apenas um conjunto com todas as situações onde

o agente pode ser potencialmente recompensado e o método, por sua vez, se encarrega de descobrir qual é a melhor combinação de features e seus respectivos sinais de recompensa que melhor definem a função de recompensa ótima. No EORP, uma feature não utilizada é removida da função de recompensa, ao invés de simplesmente atribuir a ela um sinal de recompensa que anule o seu uso. O EORP funciona desta forma porque o uso de features pode estar associado a outros custos. Por exemplo, num robô, o uso de uma feature que indica se ele bateu numa parede ou em uma porta fechada pode depender de sensores, os quais resultam em custos como o de energia e também monetário.

A.1.2 Generalidade de Aplicação

Todos os métodos existentes foram projetados para problemas bastante específicos. Isto acaba restringindo muito a gama de problemas em que cada um deles pode ser aplicado. O EORP possui uma formulação bastante geral que permite que ele seja aplicado nos mais diversos tipos de tarefa de aprendizagem por reforço (tarefas competitivas, cooperativas e mistas), desde que modificações mínimas em sua estrutura sejam feitas. Isso é possível devido a estratégia evolucionária utilizada que é independente do problema de aprendizagem e que pode lidar com configurações tanto mono quanto multiagente.

A.1.3 Escalabilidade em Problemas Multiagente

A escalabilidade é outra limitação das abordagens existentes. Grande parte das abordagens existentes para projeto de funções de recompensa, foi desenvolvida para problemas monoagente. A única que se tem conhecimento, até o momento em que este texto é preparado, que lida com problemas multiagente é o trabalho de Liu et al. (2014). A abordagem apresentada pelos autores otimiza uma função de recompensa por cada agente que aprende por reforço. Se o problema possui 10 agentes, dez funções distintas são otimizadas. Em problemas com grandes quantidades de agentes, esta estratégia pode não convergir para a solução desejada devido à baixa convergência dos métodos de otimização em espaços de busca com alta dimensionalidade. O EORP se mostrou escalável por otimizar uma única função de recompensa a qual é individualmente utilizada por cada agente que compartilha uma tarefa em comum. Sendo assim, para um determinado prob-

lema de aprendizagem, independente do número de agentes com tarefa de aprendizagem em comum, a dimensionalidade do problema de busca será sempre a mesma.

A.1.4 Esforço de Aprendizagem

O esforço de aprendizagem se refere a forma com que uma função de recompensa ótima é avaliada. No espaço de funções de recompensa pode conter múltiplas funções de recompensa que produzem um mesmo comportamento desejado, mas que diferem no esforço de aprendizagem exigido para se chegar em tal comportamento. Na nossa abordagem, durante a avaliação de uma função de recompensa, levamos em consideração o balanço entre qualidade de comportamento aprendido e esforço de aprendizagem gasto para atingir tal comportamento. Portanto, as funções de recompensa encontradas através do EORP buscam produzir o melhor comportamento no melhor (menor) esforço de aprendizagem.

A.2 Publicações

As contribuições científicas que levaram a esta tese são:

GRUNITZKI, R.; SILVA, B. C. da; BAZZAN, A. L. C. A flexible approach for designing optimal reward functions. In: DAS, S. et al. (Ed.). **Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)**. São Paulo: IFAAMAS, 2017. p. 1559–1560. Disponível em: <http://ifaamas.org/Proceedings/aamas2017/pdfs/p1559.pdf>.

GRUNITZKI, R.; SILVA, B. C. da; BAZZAN, A. L. C. Towards designing optimal reward functions in multi-agent reinforcement learning problems. In: **Proc. of the 2018 International Joint Conference on Neural Networks (IJCNN 2018)**. Rio de Janeiro: [s.n.], 2018.

Em Grunitzki, Silva and Bazzan (2017), nós apresentamos o EORP e o avaliamos na variação multiagente de dois agentes do problema de transporte coordenado de objetos. A análise no problema de alocação de tráfego, o qual possui milhares de agentes, bem como a comparação com um método para projeto de função de recompensa em problemas

multiagente, chamado difference rewards é apresentada em Grunitzki, Silva and Bazzan (2018).

Além destes trabalhos que são diretamente no tema do EORP, outras pesquisas também foram desenvolvidas ao longo da tese. São elas:

GRUNITZKI, R.; BAZZAN, A. L. C. Comparing two multiagent reinforcement learning approaches for the traffic assignment problem. In: **Intelligent Systems (BRACIS), 2017 Brazilian Conference on**. [S.l.: s.n.], 2017.

BAZZAN, A. L. C.; GRUNITZKI, R. A multiagent reinforcement learning approach to en-route trip building. In: **2016 International Joint Conference on Neural Networks (IJCNN)**. [S.l.: s.n.], 2016. p. 5288–5295.

GRUNITZKI, R.; BAZZAN, A. L. C. Combining car-to-infrastructure communication and multi-agent reinforcement learning in route choice. In: BAZZAN, A. L. C. et al. (Ed.). **Proceedings of the Ninth Workshop on Agents in Traffic and Transportation (ATT-2016)**. New York: CEUR-WS.org, 2016. ISSN 1613-0073. Disponível em: <http://ceur-ws.org/Vol-1678/paper12.pdf>.

GRUNITZKI, R.; RAMOS, G. d. O.; BAZZAN, A. L. C. Uma ferramenta para alocação de tráfego e aprendizagem de rotas em redes viárias. In: **Anais do XXVIII Congresso de Pequisa e Ensino em Transportes (ANPET 2014)**. [s.n.], 2014. ISBN 978-85-87893-17-8. Disponível em: <www.inf.ufrgs.br/maslab/pergamus/pubs/grunitzki+2014-anpet.pdf>.

RAMOS, G. de. O.; GRUNITZKI, R. An improved learning automata approach for the route choice problem. In: KOCH, F.; MENEGUZZI, F.; LAKKARAJU, K. (Ed.). **Agent Technology for Intelligent Mobile Services and Smart Societies**. [S.l.]: Springer Berlin Heidelberg, 2015, (Communications in Computer and Information Science, v. 498). p. 56–67. ISBN 978-3-662-46240-9.

RAMOS, G. de. O.; GRUNITZKI, R.; BAZZAN, A. L. C. On improving route choice through learning automata. In: **Proceedings of the Fifth International Workshop on Collaborative Agents – Research & Development (CARE 2014)**. [s.n.], 2014. p. 1–12. Disponível em: <http://www.inf.ufrgs.br/maslab/pergamus/pubs/Ramos+2014care.pdf>.

GRUNITZKI, R.; RAMOS, G. d. O.; BAZZAN, A. L. C. Individual versus difference rewards on reinforcement learning for route choice. In: **Intelligent Systems (BRACIS), 2014 Brazilian Conference on**. [S.l.: s.n.], 2014. p. 253–258.

Todos estes trabalhos estão, de alguma forma, relacionados ao tema de pesquisa desta tese. Em Grunitzki, Ramos and Bazzan (2014a), nós investigamos o uso e MARL e difference rewards no problema de alocação de tráfego. O uso de learning automata para lidar com este mesmo problema de aprendizagem é endereçado em Ramos, Grunitzki and Bazzan (2014), Ramos and Grunitzki (2015). Um simulador microscópico para lidar com o problema de alocação de tráfego em uma perspectiva multiagente é apresentado em Grunitzki, Ramos and Bazzan (2014b). Em Grunitzki and Bazzan (2016), nós investigamos o uso de comunicação car-to-car em MARL para o problema de alocação de tráfego. Uma estratégia na qual os agentes constroem a sua rota ao longo da sua viagem é investigada em Bazzan and Grunitzki (2016). Esta mesma estratégia é comparada com uma segunda estratégia, na qual os agentes aprendem a selecionar a melhor rota a partir de um conjunto de rotas pré-computadas, em Grunitzki and Bazzan (2017).